

AD-A031 625

YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE  
THE METANOVEL: WRITING STORIES BY COMPUTER.(U)  
SEP 76 J R MEEHAN

F/G 9/2

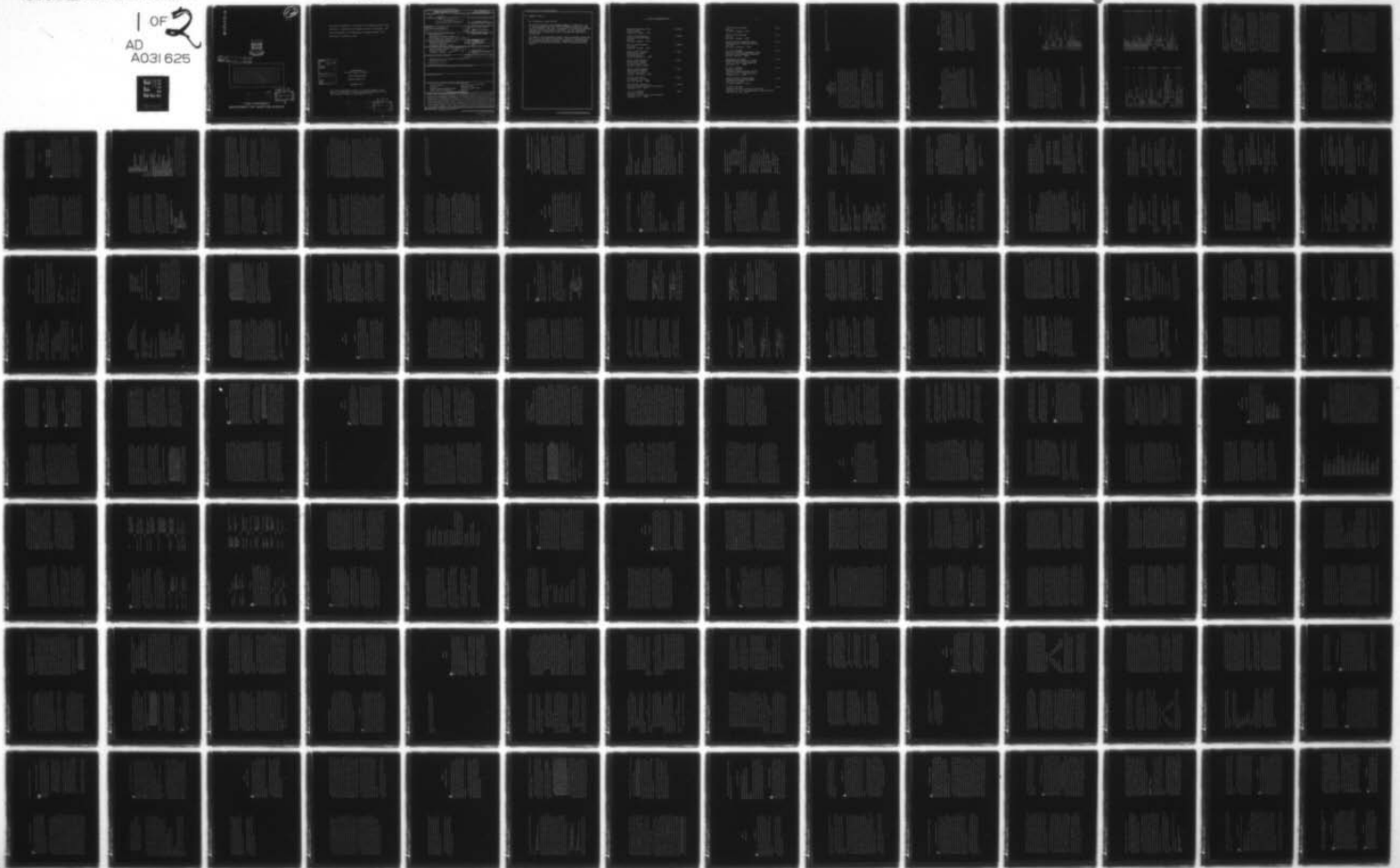
N00014-75-C-1111

UNCLASSIFIED

RR-74

NL

1 OF 2  
AD  
A031 625



SIFTED

1

OF



AD

A031 625



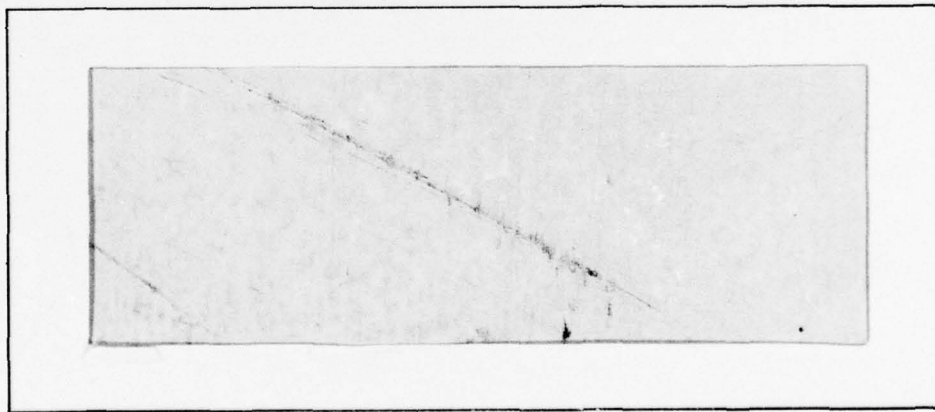


ADA031625

12  
N9



COPY AVAILABLE TO DDC DOES NOT  
PERMIT FULLY LEGIBLE PRODUCTION



DISTRIBUTION STATEMENT A  
Approved for public release;  
Distribution Unlimited

DDC  
RECORDED  
NOV 5 1976  
B

YALE UNIVERSITY  
DEPARTMENT OF COMPUTER SCIENCE

This work was presented to the faculty of the Graduate School of Yale University in candidacy for the degree of Doctor of Philosophy. The author is presently in the Department of Computer Science at the University of California, Irvine

ACCESSION for	
NTIS	White Section <input checked="" type="checkbox"/>
DDC	Buff Section <input type="checkbox"/>
UNANNOUNCED	<input type="checkbox"/>
JUSTIFICATION .....	
BY .....	
DISTRIBUTION/AVAILABILITY CODES	
Dist.	AVAIL. and/or SPECIAL
A	

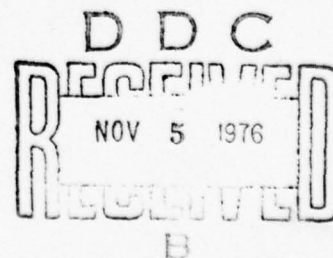
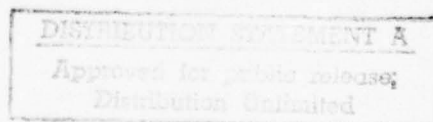
The Metanovel:  
Writing Stories by Computer

James Richard Meehan

Research Report #74

September 1976

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored under the Office of Naval Research under contract N00014-75-C-1111.



REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER #74	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Metanovel: Writing Stories by Computer.		5. TYPE OF REPORT & PERIOD COVERED Technical Report
7. AUTHOR(s) James Richard Meehan		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Yale University Department of Computer Science 10 Hillhouse Ave., New Haven, Conn. 06520		8. CONTRACT OR GRANT NUMBER(s) N00014-75-C-1111
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Office of Naval Research Information Systems Arlington, Virginia 22217		12. REPORT DATE September 1976
		13. NUMBER OF PAGES 237
		15. SECURITY CLASS. (of this report) Unclassified
16. DISTRIBUTION STATEMENT (of this Report) Distribution of this report is unlimited		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Story Generation Planning Natural Language Understanding Computational Linguistics Representation of Knowledge Problem Solving Artificial Intelligence Semantics Conceptual Dependency		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) People draw on many diverse sources of real-world knowledge in order to make up stories, including knowledge of the physical world; rules of social behavior and relationships; techniques for solving everyday problems such as transportation, acquisition of objects, and acquisition of information; knowledge about physical needs such as hunger and thirst; knowledge about stories their organization and contents; knowledge about planning behavior and the relationships between kinds of goals; and knowledge about expressing a story in a natural language. This thesis describes a computer program which uses		

DD FORM 1473

1 JAN 73

EDITION OF NOV 65 IS OBSOLETE  
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

407051



20. ABSTRACT (cont'd.)

all information to write stories.

The areas of knowledge, called problem domains, are defined by a set of representational primitives, a set of problems expressed in terms of those primitives, and a set of procedures for solving those problems. These may vary from one domain to the next. All this specialized knowledge must be integrated in order to accomplish a task such as storytelling.

The program, called TALE-SPIN, produces stories in English, interacting with the user, who specifies characters, personality characteristics, and relationships between characters. Operating in a different mode, the program can make those decisions in order to produce Aesop-like fables.

-- OFFICIAL DISTRIBUTION LIST --

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 copies
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 copies
Office of Naval Research Code 102IP Arlington, Virginia 22217	6 copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 copy
Office of Naval Research Branch Office, Pasadena 1030 East Green Street Pasadena, California 91106	1 copy
New York Area Office 715 Broadway - 5th Floor New York, New York 10003	1 copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 copy

Office of Naval Research Code 455 Arlington, Virginia 22217	1 copy
Office of Naval Research Code 458 Arlington, Virginia 22217	1 copy
Naval Electronics Laboratory Center Advanced Software Technology Division Code 5200 San Diego, California 92152	1 copy
Mr. E. H. Gleissner Naval Ship Research & Development Center Computation and Mathematics Department Bethesda, Maryland 20084	1 copy
Captain Grace M. Hopper NAICOM/MIS Planning Branch (OP-916D) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Mr. Kin B. Thompson Technical Director Information Systems Division (OP-91T) Office of Chief of Naval Operations Washington, D.C. 20350	1 copy
Advanced Research Projects Agency Information Processing Techniques 1400 Wilson Boulevard Arlington, Virginia 22209	1 copy
Professor Omar Wing Columbia University in the City of New York Department of Electrical Engineering & Computer Science New York, New York 10027	1 copy

# ABSTRACT

## The Metanovel: Writing Stories by Computer

James Richard Meehan

Yale University 1976

People draw on many diverse sources of real-world knowledge in order to make up stories, including: knowledge of the physical world; rules of social behavior and relationships; techniques for solving everyday problems such as transportation, acquisition of objects, and acquisition of information; knowledge about physical needs such as hunger and thirst; knowledge about stories, their organization and contents; knowledge about planning behavior and the relationships between kinds of goals; and knowledge about expressing a story in a natural language. This thesis describes a computer program which uses all this information to write stories.

The areas of knowledge, called problem domains, are defined by a set of representational primitives, a set of problems expressed in terms of those primitives, and a set of procedures for solving those problems. These may vary from one domain to the next. All this specialized knowledge must be integrated in order to accomplish a task such as storytelling.

The program, called TALE-SPIN, produces stories in English, interacting with the user, who specifies characters, personality characteristics, and relationships between characters. Operating in a

different mode, the program can make those decisions in order to produce  
Aesop-like fables.



## PREFACE

**T**he term metanovel, and the idea, are due to Alan Perlis. A metanovel is a computer program that tells stories that only a computer could tell, stories of such complexity of detail that only a computer could handle, stories with more flexibility -- even reversibility -- of events and characters than a human could manage. A metanovel time-sharing system tells a story to many people at once, no two of whom read the same thing, because they have each expressed different interests in the events and characters they want to hear about, and because they may each desire a *different style of storytelling*. And yet, among all these readers, there is but one story -- the Metanovel itself -- and each reader is only following those threads of the story that interest him.

The idea is intriguing and has been a constant, if distant, vision throughout the course of this work. If I've been successful, we're a little closer than we were.

#### ACKNOWLEDGEMENTS

I'd like to thank my advisor, Professor Roger Schank, for spending the time and effort to keep me on the right track. It was no mean feat to lure me away from the simple pleasures of systems programming to the challenges of natural language processing, where solving problems, it seemed to me, was like fighting the Hydra. I was right, but I didn't know it would be so much fun.

Professor Alan Perlis suggested this topic to me the very first day I met him, and he helped me get started on it. For that, and for sharing his perspective on computers, computer scientists, society, and of course, stories, I owe him a special debt of gratitude.

For having taught me most of what I know about programming, I rewarded Professors Ned Irons and Larry Snyder by having them on my thesis committee. Perhaps they'll know better next time. For the sake of their future students, I hope not.

Many of the ideas in this thesis come from the Natural Language Processing Seminar, which is a euphemism for a weekly battle of ideas, intense, but good-natured and often hilarious. One of our favorite guest speaker/victims is Psychology Professor Robert Abelson, who totally revises a theory the day you finish implementing it. My thanks also go to all the other members of the seminar, especially to Dr. Chris Riesbeck, punster, parsemaster, and class-A cognitive mechanic.

If you find a sentence in the thesis which strikes you as well written, or if you find only a few sentences at most which strike you as horribly written, then I've succeeded in using what I learned from Mary-Claire van Leunen's seminar in technical writing and her indispensable, witty book, Scholarly Prose [32].

Of course, this thesis would never have been completed without the personal support of my friends. I thank them, one and all.

---

This work was supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract N00014-75-C-1111.

## TABLE OF CONTENTS

### ABSTRACT

PREFACE . . . . .	ii
ACKNOWLEDGEMENTS . . . . .	iii
TABLE OF CONTENTS . . . . .	v

### CHAPTER 1 THE BEGINNING

1.1 A computer program that makes up stories . . . . .	1
1.2 Philosophy . . . . .	4
1.3 Other work . . . . .	6
1.4 Organization of the thesis . . . . .	9

### CHAPTER 2 EXAMPLES FROM TALE-SPIN

2.1 Program overview . . . . .	15
2.2 A detailed example . . . . .	17
2.3 Other examples . . . . .	36

### CHAPTER 3 PLANNING STRUCTURES

3.1 Introduction . . . . .	39
3.2 DELTA-PROX . . . . .	44
3.3 DELTA-NEG-PROX . . . . .	48
3.4 TELL and DELTA-KNOW . . . . .	49
3.5 Digression: special inferences in communication . . . . .	50
3.6 DELTA-CONTROL . . . . .	52
3.7 The PERSUADE package . . . . .	55
3.8 SIGMA-HUNGER . . . . .	58
3.9 SIGMA-THIRST . . . . .	59
3.10 SIGMA-REST . . . . .	59
3.11 SIGMA-SEX . . . . .	60
3.12 DO-MTRANS . . . . .	60
3.13 DO-PTRANS . . . . .	62

3.14 Relationship states . . . . .	62
3.15 Personality states . . . . .	66
CHAPTER 4       THEORY OF PLANS	68
CHAPTER 5       THEORY OF GOALS	
5.1 Goal hierarchy . . . . .	77
5.2 Cost calculus . . . . .	82
CHAPTER 6       INFERENCES AND MEMORY	
6.1 Kinds of inferences . . . . .	86
6.2 Consequences . . . . .	91
6.3 Reactions . . . . .	93
6.4 Organization of memory . . . . .	95
6.5 Cheating . . . . .	100
CHAPTER 7       THEORY OF STORIES	
7.1 Introduction . . . . .	102
7.2 Story levels . . . . .	103
7.3 The two methods of telling stories . . . . .	107
7.4 The Aesop-fable generator . . . . .	108
7.5 Coherence . . . . .	113
7.6 Interest . . . . .	114
7.7 Style . . . . .	119
7.8 Complexity . . . . .	123
CHAPTER 8       MIS-SPUN TALES	126
CHAPTER 9       MAPS AND BLUEPRINTS	
9.1 Maps . . . . .	136
9.2 Blueprints . . . . .	143
9.2.1 What's an office building, really? . . . . .	144
9.2.2 Didn't I just walk out of here? . . . . .	145
9.2.3 What do you mean, do I want a new mountain? . . . .	146
9.2.4 What do you mean, what kind of nest do I want? . . .	148
CHAPTER 10      SCHEDULING EVENTS IN A STORYTELLER	150
CHAPTER 11      DETAILS, DETAILS	
11.1 Introduction . . . . .	157
11.2 A very detailed example . . . . .	158
11.3 Digression: minimal environment . . . . .	158
11.4 Creating a bear . . . . .	160
11.5 Digression: random choices vs. theory . . . . .	161
11.6 Creating a bear, continued . . . . .	162
11.7 Creating physical locations . . . . .	164

11.8	The first problem	167
11.9	SIGMA-HUNGER	168
11.10	DELTA-CONTROL	169
11.11	DELTA-KNOW	169
11.12	Relationships	170
11.13	PERSUADE	173
11.14	TELL	174
11.15	DELTA-PROX	174
11.16	How close is close?	175
11.17	DELTA-LINK	176
11.18	Digression: noticing	178
11.19	DO-MTRANS	180
11.20	REQUEST	181
11.21	Back in DELTA-CONTROL	185
11.22	Persuading again	186
11.23	The lie detected	187
11.24	BARGAIN	187
11.25	PROMISE	188
11.26	The demon wakes	191
11.27	THREATEN	192
11.28	The fall of Arthur	193
11.29	Programming details	
	11.29.1 Data vs. procedure	194
	11.29.2 The syntax of CD	195
	11.29.3 PATH and XPLD	196
CHAPTER 12 TELLING A STORY -- IN ENGLISH		
12.1	The generator	199
12.1.1	Verbs (acts)	200
12.1.2	States	204
12.1.3	Causals	204
12.1.4	Verb modifiers	206
12.1.5	Articles	206
12.2	Dialogue	207
CHAPTER 13 THE END		
13.1	Using people as models of storytellers	211
13.2	Extensions	212
13.3	Stories	219
13.4	World knowledge	222
13.5	The metanovel revisited	222
13.6	Planning and problem solving	224
APPENDIX: More examples		227
REFERENCES		234

## CHAPTER 1

### THE BEGINNING

#### 1.1 A COMPUTER PROGRAM THAT MAKES UP STORIES

**W**hat do you need to know in order to make up a story? This is the central question addressed by this thesis. Our method is to separate that knowledge into individual areas, to build a computer program which models each area, and to put all these programs together under the direction of one master program. The emphasis is on knowledge, not on style of expression, nor even on the structure of stories themselves, for there are simpler ways to force computers to "tell" stories. The simplest method is to give the computer the entire story and ask it to tell it back to you. From that extreme you might proceed in the direction of telling the computer fewer explicit things and more general things. You might give the computer some fancy notation for representing what happens in the story and ask it to express it us in English. You might give it some canned plots and ask it to fill in the names and add a little color, at random perhaps. But far more



interesting, and far more significant to the field of Artificial Intelligence, is the the question: What do humans need to know in order to make up stories?

The computer program I will discuss contains a model of the physical world and the world of human behavior. It simulates characters who have motives, emotions, and relationships with other characters. It does this by using a great deal of information about those motives, emotions, and relationships. It knows about stories, what's interesting, what's coherent. It tells us what happens as the simulation proceeds.

The particular world model I use consists of humans and talking animals. Their principal motivations stem from physical needs such as hunger. The physical world has mountains, caves, forests, houses, and the like. It is not a fixed world. Individual parts are created as required by the story. It does not model social institutions.

The name of the program is TALE-SPIN. My chief concern is modeling the world by integrating many theories and sources of information into one program. The overall structure of TALE-SPIN is a simulation of rational behavior by the characters in the model. The simulator has three active components, shown in Figure 1. There is a problem solver which, given a goal, produces other goals (subgoals) and actual events. There is an assertion mechanism which takes an event and adds it to the world model, the static description of each knowledge area (what the physical world looks like at that instant, what social relationships





## 1.2 PHILOSOPHY

**T**ALE-SPIN is intended to model people. For example, in designing the model of the physical world, I wanted a close correspondence to the way in which humans think about the physical world. The result may not be the most efficient model for a computer. In figuring out the distance from London to Paris, people resort to all sorts of calculations, such as relating that distance to the time it takes to make the journey by plane. A simple computer model for that problem might include assignments of X-Y coordinates to each city, and the distance calculation would be simple: Take the square root of the sum of the squares of the differences in coordinates. The approach here is not to use the square root method, but rather to figure out what it is that people do.

We have several goals in all of this. We're interested in studying people, not computers, so we build models which correspond to the way in which we think people behave. We also know that people are capable of solving an astonishing array of kinds of problems, and with remarkable efficiency. They must be doing something right, so if we want the computer to solve problems as well as people do, we should try to get the computer to imitate people. Finally, we wish to be able to communicate with a computer in our own terms, regardless of the subject matter, whether it's payrolls or politics, and just as every hearer must have a model of the speaker in order to understand him, so too must the computer have a model of how people think, since that determines what

people say and do.

Modeling humans also enables us to estimate where the boundaries are between one area of knowledge and another. The fact that the computer can do numerical operations quickly *does not mean* that our model should do all its arithmetic with one procedure, because people don't handle all numerical problems in the same way. In general, I am not terribly concerned with programming issues such as data structures and code structures. If I know an efficient way to implement something, and it's not inconsistent with the theory, I'll use it. But questions of overall control structure are certainly premature, and any commitment to some fancy way of programming is not only expensive in terms of hours, it's also likely to get in the way as we expand the capabilities of the model. To put it another way, there's no point to building the assembly line to manufacture cars efficiently until you've finished designing the car.

There is always a question of how much knowledge to put into a program. The goal of organizing and representing human knowledge is very broad and, without a specific task in mind, very vague. TALE-SPIN's task is to write stories, so the criterion for adding new information is that it enable the program to produce new kinds of stories. In early versions of TALE-SPIN [17], some decisions had to be made by the reader because the program didn't know enough. Those decisions have, by and large, been removed. The questions that remain concern issues outside the story itself, more at the author's level than

the reader's, such as the choice of characters. To use an analogy, consider a program which adds any two integers. The program could be both efficient and accurate, but it would not in any way help you decide which integers to add. That's a decision outside the program itself.

### 1.3 OTHER WORK

"FLASH THERE IS ANOTHER MECHANISM"

-- Colossus,

in Colossus: The Forbin Project

by D. F. Jones

**A**t the University of Wisconsin, Sheldon Klein and his students have a system which also produces stories [2,14,15]. Their goal is to build a very general simulation system. As an example of something to simulate, they choose the production of stories, using the theories of Vladimir Propp [20], a Russian folklorist who analyzed a class of Russian folk tales and developed a notation to represent their common structural features. Klein reverses Propp's analytic system, turning it into a generative system for producing stories which are supposed to look like examples of that class of Russian folk tales.

Klein's system includes a new programming language called MESSY in which he specifies the rules of the simulation, and he is very concerned with such issues as writing speed. His "Propp model" generates stories at an average speed of 128 words per second.

Obviously, Klein and I have different goals in mind. The function of his system is to make a simulation as efficient as possible. It is no more concerned with the accuracy of the model which is being simulated than a Xerox machine is concerned with the content of the material it is copying. My prime concern in TALE-SPIN is the model itself, not its simulator.

The reason I emphasize the model is that I have found that only after studying the model is it clear what the control structure of the simulator should be. Klein's system can randomly choose what happens next. That's the last thing I want TALE-SPIN to do; it models what should happen next.

TALE-SPIN has general techniques for solving problems. It uses the Conceptual Dependency representation of meaning [25] and the general solutions are expressed as combinations of primitive states and acts. Klein's system has no primitives -- or, if you like, arbitrarily many primitives -- and the simulation rules are coded in explicit detail. Here is an example from Klein's "murder mystery model."

```
%
%
%           HITTING SOMEONE OVER THE HEAD WITH A HEAVY
%           OBJECT FOR BLACKMAILING YOU.
%
SLOOP K9:  H.PICK(HEAVYOBJ);
$RULE:    *MOVE H TO EVIDENCE.
          *MOVE H TO WEAPON.
          *MOVE FEAR TO MOTIVE.
          V BLACKMAIL K.
          *INSERT (K MADAT V)(MADAT VERY).
          *INSERT (K AFRAID)(AFRAID OF V).
          *INSERT (K DECIDE)(DECIDE KILL V).
          DAY IS SUNDAY.
          TIME IS DAWN.
          K GETUP.
```

```

K GOTO HALL.
MX QQ = 1.
HALL DARK.
K HIDE NO.
K HAVE H.
*INSERT (V AWAKEN)(AWAKEN EARLY).
*INSERT (V EARLY)(EARLY USUALLY).
V GOFOR WALK.
K WAITFOR V.
K SURPRISE V.
*INSERT (K HIT V)(HIT WITH H).
*INSERT (V GROAN)(GROAN WEAKLY).
V DIE;
$RULE: *INSERT (K REMOVE FPRINTS)(FPRINTS ON H);
      : (K IQ)/150;
$RULE: ($ENDGROUP)
      K RETURN TO BEDROOM;
$ENDLOOP;

```

That produced the following murder scene:

```

DR. BARTHOLOMEW HUME BLACKMAILED EDWARD.
EDWARD WAS AFRAID OF DR. HUME.
LORD EDWARD DECIDED TO KILL DR. BARTHOLOMEW HUME.
THE DAY WAS SUNDAY.
THE TIME WAS THE SUNRISE.
LORD EDWARD GOT UP.
LORD EDWARD WENT TO THE DARK CORRIDOR.
LORD EDWARD HID.
EDWARD HAD A CANDLE HOLDER.
DR. BARTHOLOMEW HUME AWAKENED EARLY.
DR. BARTHOLOMEW HUME WAS USUALLY EARLY.
DR. HUME WENT FOR THE WALK.
EDWARD WAITED FOR HUME.
LORD EDWARD SURPRISED HUME.
EDWARD HIT DR. BARTHOLOMEW HUME WITH THE CANDLE HOLDER.
DR. BARTHOLOMEW HUME GROANED WEAKLY.
DR. HUME DIED.
EDWARD RETURNED TO THE BEDROOM.

```

If you look at the code and the story together, line by line, you can see that there's no logic in the model -- the code is completely explicit. In no way does the program understand, for example, the meaning of the idea "surprise." The code said "K SURPRISE V" so the program printed "LORD EDWARD SURPRISED HUME." There's nothing in there

about the fact that Hume was not expecting something, that Edward wanted Hume not to know about something, that Edward did something which caused Hume to find out about something very quickly, and so on. These are what the idea "surprise" means, and it doesn't know that. Why did Hume die? Not because he was hurt: That's an inference you and I made when we read that he was struck with the candlestick. He died because the code said "V DIE."

This isn't a good model of either stories or storytelling because there's no implicit causality in the stories: nothing that says why one thing implies another, nothing that says what people need, nothing that says why people make the decisions they do, nothing that explains people's reactions to one another, nothing that represents the complexities of the physical world. The purpose of TALE-SPIN is to address exactly these issues.

#### 1.4 ORGANIZATION OF THE THESIS

Chapter 2, "Examples from TALE-SPIN," gives an overview of TALE-SPIN and then presents an annotated story: output from the program interspersed with some explanatory comments. Section 2.3 contains additional, unannotated stories generated by TALE-SPIN.

Chapter 3 describes planning structures, based on the work of Roger Schank and Robert Abelson [1,28,29]. They designed them as a means of understanding narratives about problem-solving behavior. I use them to



explain the methods people actually use in solving everyday problems. THE OUTPUT FROM TALE-SPIN IS A TRACE THROUGH GOAL-DIRECTED PROCEDURES. For example, the program for solving the "problem" of getting from one place to another produces, as a by-product, an account of the journey and all the preparations for the journey, including whatever dealings with other people may have been necessary.

Chapter 4, "Theory of Plans," discusses the difference between this approach to problem-solving and the traditional approaches used by Newell and Simon (GPS), Fikes, Hart, Nilsson, and Sacerdoti (STRIPS), and Fahlman (BUILD).

Planning is always done within a context of goals. Chapter 5, "Theory of Goals," describes two goal processes, achievement and preservation, and a cost calculus which helps us decide which goals to pursue and which to abandon.

Most of what happens in the simulation is not directly controlled, but rather works as side-effects. If the simulator specifies that John walks up to Mary, a side-effect will be that she notices him; the simulator doesn't have to say that. Having noticed him, she may start speaking to him. This, too, is not directly controlled by the simulator. These side-effects are actually consequences of the controlled events. Other inferences allow the characters themselves to make involved decisions, in bargaining sessions, for example. The inferences in TALE-SPIN are described in Chapter 6.



Each of the characters has a separately maintained memory, and the storyteller does, too. John's beliefs may not coincide with Mary's, and neither may coincide with the truth (the storyteller's beliefs). Chapter 6 also discusses the organization and use of memory in TALE-SPIN.

In Chapter 7, we take a look at stories and explore the notion that stories are about problems of one kind or another. Two general approaches to storytelling and their implementations in TALE-SPIN are discussed, along with a specific example: How to write an Aesop fable.

Storytelling was chosen as a vehicle for modeling the world, and it has proved to be an excellent source of problems, not so many that it was impossible to begin and get some results, nor so few that exploring new ones didn't add to the quality of the stories produced. To be sure, there are many areas of human knowledge that are of minimal use to storytellers, principally those areas of "specialized" knowledge which would have to be explained to a reader before he could understand their use. But even the simplest texts require surprising amounts of information, things we take for granted. Computers, of course, take absolutely nothing for granted, except perhaps that their programmers know exactly what they're doing. During the programming of TALE-SPIN, many of my tacit assumptions were revealed in the form of absurd stories; that is, because I hadn't specified to the machine how such-and-such a thing worked, the computer omitted it, and the resulting story contained bizarre behavior by one or more characters. Chapter 8,

"Mis-spun Tales," is a collection of just such oddities.

People spend much of their waking life involved in planful behavior of one type or another, ranging from the very high-level plans to the very low-level plans, with the emphasis on the low-level. Characters in stories, however, rarely have their low-level planning activity described -- it's boring, so why write about it? "John chewed the steak." Ho hum. Consequently, a storytelling program will have to spend more of its time describing higher-level planning behavior. TALE-SPIN, in fact, spends most of its time having the characters use general techniques to figure out solutions to new problems.

But one particular low-level problem which required a detailed solution was the representation of the physical world -- how people calculate routes and distances. One of the principal goals of that study was psychological credibility, avoiding, for instance, square roots in calculating distances. Another goal was a uniform representation of geographical information, so that we can process it in one general way, whether the story takes place in a city or in the country. A third goal was a description of abstract physical objects, not only specific ones, so that, for example, when someone talks about walking through a building, you don't have to know the particular building in order to follow his description. You should be able to use your abstract model of a building to fill in details as you go along. I call the abstract models "blueprints" and the specific models "maps." They are discussed in Chapter 9.

Generation of text is often referred to as the "what to say" problem, or, in considering style, the "how to say" problem. Chapter 10 discusses what I call the "when to say" problem, or in general, the problem of scheduling events in a story in the same way that a human storyteller might.

Chapter 11, "Details, Details," follows the production of a TALE-SPIN story as did Chapter 2, but now in much more detail. Chapters 3 through 10 presented the individual pieces of TALE-SPIN and their theoretical backgrounds. Chapter 11 shows the whole system in action, how individual characters are created, how the various planning mechanisms interrelate, how inferences are applied, on and on.

At the end of Chapter 11, I describe some of the programming techniques used in TALE-SPIN. I firmly believe that programming details are rarely important in understanding AI systems, that you can do it all in FORTRAN anyway, horresco referens, so there are no lines of code anywhere but in section 11.29. On the other hand, there may be readers who find those details to be of some interest. But the important things in this program are the ideas behind the code, not the code itself. If there were some new code structure which was developed because it not only proved useful but also corresponded closely to a psychological model, then that code structure would deserve explication.

Chapter 12 returns to the "what to say" and "how to say" issues. The English generator used in TALE-SPIN and described in Chapter 12 is based on Goldman's BABEL.

Finally, Chapter 13 contains some general comments on the work, the inevitable suggestions for future work, and a parting view of this approach to AI.

## CHAPTER 2

### EXAMPLES FROM TALE-SPIN

#### 2.1 PROGRAM OVERVIEW

**T**ALE-SPIN makes up stories by simulating a world, giving some characters some goals, and telling us what happens. The reader gets to supply much of the information about the initial state of the world, such as the choice of characters and the relationships between one character and another. There are three modes in which the program runs. In mode 1, almost everything which happens appears as a part of the story. If John and Mary are sitting in a room, talking, and John leaves, the program says, among other things, that Mary notices that John has left. A human storyteller would say that only if there were some consequence of her noticing. Example:

Mary noticed that John had left. She rushed back to catch the end of her soap opera. John hated soap operas.

The implication here is that she would not watch the show if John were around. Another example:

Mary wanted to remind John about their plans for dinner. She noticed that he had left. "Oh well," she thought, "I'll call him later."

John's departure has influenced how Mary will communicate with him.

Unusual events are also expressed in stories.

Mary went in the kitchen to get John his drink. When she came back, she noticed he had left.

But in mode 1, "verbose" mode, TALE-SPIN says what happens without regard to its relevance to the story as a whole. Mode 1 serves primarily to illustrate all the things which are actually happening in the simulated world, and watching it can be amusing.

In mode 2, the storyteller carries on the same question/answer dialogue with the reader, but the story comes out all at once, at the end. Mode 2 models the process of thinking your way through a story first, and then telling it. It is easier to decide what things are worth saying, because from hindsight, as it were, the storyteller knows whether an event had any consequences and would therefore be worth saying.

In mode 3, the program uses a predefined set of morals or "points" in writing the story. These can help specify what things need to be included in the world model. If the point we choose for a story is "Never trust flatterers," then the storyteller knows that the story must have at least a character who flatters, a character who is flattered, and a situation of losing which is caused by the flattery. If we want Bill to trick Henry out of some possession, then we have already defined

certain things about their individual personalities and about their relationship. In contrast, modes 1 and 2 begin with a completely empty world.

## 2.2 A DETAILED EXAMPLE

Here is a mode-1 story written by TALE-SPIN. Upper case output is from the program, either as part of the story ("JOE BEAR ATE THE HONEY"), or, in brackets, as comments on new items which have been added to the world model ("[ CREATING SOME NEW BERRIES ]"), or as part of the question/answer dialogue ("HOW HUNGRY IS IRVING BIRD?"). Our answers to these questions are typed when an asterisk (\*) appears. The program begins:

PLEASE CHOOSE ONE OF THE FOLLOWING:

- 1: VERBOSE MODE
- 2: NOT-SO-VERBOSE MODE
- 3: PREPACKAGED PLOT

\*1

We ask for mode 1.

\*\*\*\*\* WELCOME TO TALE-SPIN \*\*\*\*\*

CHOOSE ANY OF THE FOLLOWING CHARACTERS FOR THE STORY:

(BEAR BEE BOY GIRL FOX CROW ANT HEN LION DOG WOLF MOUSE

CAT GOAT CANARY)

We choose some initial characters for the story. We could choose a fox, three cats and a dog, whatever we like.

\*(BEAR BEE BOY CANARY)

We start with four characters.

[ CREATING JOHN BEAR ]

Except for the choices of BOY or GIRL, the sex of each character is decided at random, with 50-50 odds. First names are chosen at random, as are last names for humans.

[ CREATING A NEW CAVE (\*CAVE\*0) ]

When the program creates a character, it also creates a minimal environment for the character, including at least a home which is appropriate for the character. Bears live in caves, birds in nests, and so on. (\*CAVE\*0 is the program's name for this cave.) The idea of a minimal environment says only that we do not imagine characters existing in a vacuum, that some other things come along with the notion "bear." We're not telling a creation myth; we're starting a story in a world which is supposed to exist already. For more details on minimal environments, see Chapter 11.

[ CREATING A NEW MOUNTAIN (\*MOUNTAIN\*0) ]

The cave itself also needs an environment. In the representation of the



physical world which the program uses, mountains, meadows, and valleys are at the highest level and therefore require no environment. This representation is discussed in detail in Chapter 9.

#### JOHN BEAR IS AT A CAVE.

When a character is created, we arbitrarily decide to place him in his home, as good a starting place as any. "John Bear is at a cave" is the first "fact" of the story, and it is stored in memory where it can be accessed later on. When facts are added to memory, the program checks whether there are any inferences that can now be made that would produce more events for the story. This sentence has no such inferences; we'll soon see one that does. An inference that actually is produced from this sentence is that John Bear knows (believes, thinks) that he's at the cave. That inference is also added to memory but is not included in this output.

[ CREATING IRVING BEE ]

[ CREATING SOME NEW HONEY (\*HONEY\*0) ]

[ CREATING A NEW BEEHIVE (\*BEEHIVE\*0) ]

The environment for a bee includes more than just a beehive. In stories, at least, the association between bees and honey can be as strong as the association between farmers and crops, between bankers and money. The farmer who is paid not to grow crops is memorable precisely because this association is violated.

[ CREATING A NEW APPLE TREE (\*APPLETREE\*) ]

[ CREATING A NEW GROUND (\*GROUND\*0) ]

[ CREATING A NEW VALLEY (\*VALLEY\*0) ]

A BEEHIVE IS AT AN APPLE TREE.

IRVING BEE IS AT THE BEEHIVE.

SOME HONEY IS AT THE BEEHIVE.

IRVING BEE HAS THE HONEY.

These facts about Irving Bee are now added to memory, along with inferences, not shown here, that he knows he's at the beehive, that he knows the honey is at the beehive, and that he knows he owns the honey.

[ CREATING SAM ADAMS ]

[ CREATING A NEW HOUSE (\*HOUSE\*0) ]

[ CREATING A NEW FRONT DOOR (\*DOOR\*0) ]

[ CREATING A NEW VALLEY (\*VALLEY\*1) ]

SAM ADAMS IS AT A HOUSE.

[ CREATING A NEW NEST (\*NEST\*0) ]

[ CREATING A NEW REDWOOD TREE (\*REDWOODTREE\*) ]

[ CREATING A NEW GROUND (\*GROUND\*1) ]

[ CREATING A NEW MEADOW (\*MEADOW\*0) ]

[ CREATING WILMA CANARY ]

WILMA CANARY IS AT A NEST.

CHOOSE ANY OF THE FOLLOWING PROPS:

(BREADCRUMBS CHEESE BASEBALL)

\*NIL

In addition to characters and objects in their environment, the world may contain objects which are possessed by characters. Characters given such objects are, of course, made aware that they own them, and may use them. We didn't create any this time.

CHOOSE ANY OF THE FOLLOWING MISCELLANEOUS ITEMS:

(BERRIES FLOWER RIVER WORM)

\*(BERRIES WORM)

We can also include some items which exist in the world but are not possessions of any character.

[ CREATING SOME NEW BLUEBERRIES (\*BLUEBERRIES\*) ]

[ CREATING A NEW BUSH (\*BUSH\*0) ]

These also get an environment.

[ CREATING A NEW MEADOW (\*MEADOW\*1) ]

[ CREATING A NEW PLACE IN MEADOW (\*MEADOWPOINT\*1) ]

SOME BLUEBERRIES ARE AT A BUSH.

[ CREATING A NEW WORM (\*WORM\*0) ]

[ CREATING A NEW PATCH OF GROUND (\*PATCH\*0) ]

[ CREATING A NEW MEADOW (\*MEADOW\*2) ]

[ CREATING A NEW PLACE IN MEADOW (\*MEADOWPOINT\*2) ]

A WORM IS AT A PATCH OF GROUND.

WHO KNOWS ABOUT THE BLUEBERRIES?

1: WILMA CANARY      2: SAM ADAMS      3: IRVING BEE

4: JOHN BEAR

Somebody, possibly more than one character, gets to find out about these items. At least one character has to know about these things. Otherwise there would have been no point in creating them.

\*4

JOHN BEAR KNOWS THAT THE BLUEBERRIES ARE  
AT THE BUSH.

We let John Bear know about the berries.

HOW HUNGRY IS JOHN BEAR?

1:VERY 2:SOMEWHAT 3:NOT VERY 4:NOT AT ALL

The inferences from a given state or act are not always the same. They often depend on what things are already known, on context. When the information in the previous sentence was added to memory, inferences were made, as always. In this case, however, a new kind of inference could be made, a story-generating inference. Memory knows that bears eat berries, and it hasn't heard that John Bear ate anything recently. So an inference from the idea that John is hungry is that he will do something about it. Making this inference activates a procedure which generates a story about John's getting some food and eating it.

\*4

JOHN BEAR IS NOT AT ALL HUNGRY.

Had we said otherwise, the program would have begun the story

immediately by activating a satisfy-hunger goal for John Bear.

WHO KNOWS ABOUT THE WORM?

1: WILMA CANARY      2: SAM ADAMS      3: IRVING BEE

4: JOHN BEAR

\*2

SAM ADAMS KNOWS THAT THE WORM IS AT THE  
PATCH OF GROUND.

Since the program knows that Sam Adams, a human, doesn't eat worms, there's no story-generating inference from this as there was in the previous example. Had we said that Wilma Canary knew about this worm, we would have been asked how hungry she was.

THIS IS A STORY ABOUT ...

1: WILMA CANARY      2: SAM ADAMS      3: IRVING BEE

4: JOHN BEAR

\*2

We now choose a "main character."

HIS PROBLEM IS THAT HE IS ...

1: HUNGRY      2: THIRSTY      3: HORNY      4: TIRED

\*1

The main character gets a goal. These particular goals are called "sigma-states" and represent physical needs which must be satisfied from time to time. We choose one to begin the story. It is not essential

that the goal be achieved in order to have an interesting story, nor need this goal be the central theme of the story. It is simply the impetus for the first character to come alive.

#### SAM ADAMS IS SOMEWHAT HUNGRY.

The program asserts that Sam becomes hungry. The degree of hunger is randomly decided. It might have been "famished," for example. In the present program, it doesn't yet matter how hungry he is. As long as he is at all hungry, he will begin to look for food. The degree of hunger could be of use, however. The effect of making Sam famished instead of mildly hungry is that the solution to this problem would have additional constraints on time. That is, he would have to get some food quickly. The more important it is that he eat, the less important it is what he eats. If he were sufficiently desperate, he might steal food even if he were not normally disposed to stealing. The issue of matching goals and their constraints is discussed in Chapter 5.

#### SAM ADAMS WANTS TO GET SOME BERRIES.

The procedure for satisfying hunger (SIGMA HUNGER) is now active. What do you do when you're hungry? You get some food. Memory is checked to see whether Sam already owns any food. We haven't given him any yet, so the program chooses a food for him to get.

#### SAM ADAMS WANTS TO FIND OUT WHERE SOME BERRIES ARE.



Before you can get something, you have to know where to go to get it.  
Sam *doesn't* know that, either.

DOES SAM ADAMS LIKE WILMA CANARY?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

One way to find out information is to ask your friends, but we haven't said whether Sam has any friends. Every initial character knows that every other initial character exists, but no more than that, so the program asks about each of Sam's acquaintances until it finds a friend, that is, until the user says that they relate to each other in a certain way on a scale of affection. *There is a set of such scales which combine to describe how the characters relate.* The relationships are not symmetric. The question "Does Sam Adams like Wilma Canary?" is asked from Sam Adams's perspective. Wilma may not at all agree. She may think that Sam hates her. Various parts of these relationships are tested during the solution of a problem. If a part of the relationship is not known, the reader gets to decide. Relationships and personality characteristics are described more fully in Chapter 3.

SAM ADAMS WANTS WILMA CANARY TO TELL SAM ADAMS

WHERE SOME BERRIES ARE.

Wilma meets the "friend" criterion, so Sam now considers how to persuade her to tell him where some berries are.

DOES SAM ADAMS FEEL DECEPTIVE TOWARDS WILMA CANARY?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

DOES SAM ADAMS FEEL COMPETITIVE TOWARDS WILMA CANARY?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

One of the ways Sam can get Wilma to do something for him is simply to ask her to do it, but that requires more information about the relationship. We said that Sam feels competitive towards Wilma, so he's not going to ask her about the berries.

SAM ADAMS DECIDES THAT WILMA CANARY MIGHT  
WANT SAM ADAMS TO GIVE WILMA CANARY A WORM.

SAM ADAMS WANTS TO ASK WILMA CANARY WHETHER  
WILMA CANARY WILL TELL SAM ADAMS WHERE SOME  
BERRIES ARE IF SAM ADAMS GIVES WILMA CANARY  
A WORM.

Sam is bargaining with Wilma. He comes up with this particular idea by considering what her goals are, what the requirements of those goals are, and what things he can do towards satisfying some of those requirements. He knows that canaries need to eat and that they like to eat worms, so it's reasonable to offer to bring her a worm in exchange for the information about berries.

SAM ADAMS WANTS TO GET NEAR WILMA CANARY.

Before he can offer her anything, he has to be near her so he can talk to her.

[ CREATING A NEW VALLEY-MEADOW BORDER (\*BORDER\*0) ]

[ CREATING A NEW MEADOW-VALLEY BORDER (\*BORDER\*1) ]

The program had created Sam's house in the valley and Wilma's tree in the meadow, but now it has to create a way to get from one to the other.

SAM ADAMS WALKS FROM THE HOUSE TO THE GROUND  
BY THE REDWOOD TREE BY GOING THROUGH A  
VALLEY THROUGH A MEADOW.

SAM ADAMS IS AT THE GROUND BY THE REDWOOD  
TREE.

The journey takes place. Since Sam walks and doesn't fly, he goes to the ground by the tree, as opposed to being in the tree or in the nest with Wilma.

WILMA CANARY KNOWS THAT SAM ADAMS IS AT THE  
GROUND BY THE REDWOOD TREE.

One of the results of going to Wilma is that she notices his presence, which enables him, for instance, to talk to her.

SAM ADAMS ASKS WILMA CANARY WHETHER WILMA  
CANARY WILL TELL SAM ADAMS WHERE SOME BERRIES  
ARE IF SAM ADAMS GIVES WILMA CANARY A WORM.

He asks the question.

DOES WILMA CANARY FEEL DECEPTIVE TOWARDS SAM ADAMS?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*1

In order to know how Wilma should respond to this offer, the program needs to know something about Wilma's perception of her relationship with Sam. We make her a sneaky character.

WILMA CANARY TELLS SAM ADAMS THAT WILMA  
CANARY WILL TELL SAM ADAMS WHERE SOME BERRIES  
ARE.

She says she will, but since we made her a sneaky character, she actually has a trick in store, as we'll see later.

SAM ADAMS THINKS THAT WILMA CANARY WILL TELL  
SAM ADAMS WHERE SOME BERRIES ARE.

He believes her, so he begins to carry out his part of the bargain. When someone believes something which is known to be true, the verb "know" is used; otherwise, as in this case, we use the verb "think."

SAM ADAMS WANTS TO GET A WORM.  
SAM ADAMS WANTS TO GET NEAR THE WORM.

Ah! He remembers "the" worm which we created at the very beginning of the story.

IN CREATING A VALLEY, WE CAN MAKE UP A NEW ONE

OR USE AN OLD ONE. DO YOU WANT TO USE ANY OF THESE?

1: \*VALLEY\*0      2: \*VALLEY\*1

-- DECIDE:      \*YES

PLEASE TYPE AN INTEGER BETWEEN 1 AND 2

\*2

The part of the program which finds routes from one place to another (call it the travel agent) knows that Sam is in a valley and that the worm is in a meadow. The travel agent also knows that valleys and meadows can be adjacent, or can be separated by any number of other valleys and meadows. Having no information about the adjacency of this valley and this meadow, it asks us whether we want the intermediate areas. We say no, so it makes the valley and meadow adjacent.

[ CREATING A NEW VALLEY-MEADOW BORDER (\*BORDER\*2) ]

[ CREATING A NEW VALLEY-MEADOW BORDER (\*BORDER\*3) ]

[ CREATING A NEW MEADOW-VALLEY BORDER (\*BORDER\*4) ]

[ CREATING A NEW MEADOW-VALLEY BORDER (\*BORDER\*5) ]

SAM ADAMS WALKS FROM THE GROUND BY THE

REDWOOD TREE TO THE PATCH OF GROUND BY GOING

THROUGH THE MEADOW THROUGH THE VALLEY

THROUGH A MEADOW.

Having created the necessary connections, the program has Sam walk to the patch of ground where the worm is.

WILMA CANARY KNOWS THAT SAM ADAMS ISN'T AT

THE GROUND BY THE REDWOOD TREE.

One of the inferences about going from place to place is that the people you leave behind notice that you're gone. Should they now decide to talk to you, for example, they'll have to find you. *Wilma doesn't know* where Sam is -- he didn't say anything about that to her -- but she knows where he isn't.

SAM ADAMS TAKES THE WORM.

The worm is a completely passive character.

SAM ADAMS WANTS TO GET NEAR WILMA CANARY.

This goal is part of the bargaining operation.

SAM ADAMS WALKS FROM THE PATCH OF GROUND TO THE GROUND BY THE REDWOOD TREE BY GOING THROUGH THE MEADOW THROUGH THE VALLEY THROUGH THE MEADOW.

WILMA CANARY KNOWS THAT SAM ADAMS IS AT THE GROUND BY THE REDWOOD TREE.

The travel agent doesn't need to create anything, since a route is already known. Wilma notices Sam's return.

SAM ADAMS GIVES WILMA CANARY THE WORM.

He carries out his part of the bargain.

WILMA CANARY KNOWS THAT SAM ADAMS GAVE WILMA CANARY THE WORM.



This relates to the trick she has in store.

WILMA CANARY KNOWS THAT WILMA CANARY HAS THE WORM.

HOW HUNGRY IS WILMA CANARY?

1:VERY 2:SOMEWHAT 3:NOT VERY 4:NOT AT ALL

\*1

Again, as soon as Wilma "realizes" that she has some food, we are asked whether she's hungry. We say yes.

WILMA CANARY EATS THE WORM.

THE WORM IS GONE.

WILMA CANARY IS NOT VERY HUNGRY.

All trace of the worm in active memory is now erased. The program assumes that Wilma won't eat anything for a while. Should she see another worm soon, it will assume that she's not interested. (We're assuming that one worm was enough to satisfy her hunger.)

WILMA CANARY TELLS SAM ADAMS THAT SAM ADAMS IS NOT AT ALL SMART.

Wilma has had this in mind ever since we decided to make her a sneaky character. She had no intention of complying with his request, but was simply being nasty. She completes the trick by insulting Sam.

SAM ADAMS LIKES WILMA CANARY NOT AT ALL.

SAM ADAMS TRUSTS WILMA CANARY NOT AT ALL.

Sam understands that he has been insulted.

SAM ADAMS DECIDES THAT WILMA CANARY MIGHT WANT SAM ADAMS

TO GIVE WILMA CANARY SOME CHEESE.

But he persists. The social requirements for bargaining are not very strict.

SAM ADAMS ASKS WILMA CANARY WHETHER WILMA CANARY WILL  
TELL SAM ADAMS WHERE SOME BERRIES ARE IF SAM ADAMS GIVES  
WILMA CANARY SOME CHEESE.

Birds, supposedly, eat cheese, so Sam tries again.

WILMA CANARY TELLS SAM ADAMS THAT WILMA CANARY WILL TELL  
SAM ADAMS WHERE SOME BERRIES ARE.

You must admit, she's consistent. In fact, once we've set up the characteristics and relationships, they remain that way unless something happens in the story which changes them. For instance, Sam Adams originally liked Wilma, but when he discovered that she tricked him, he changed his mind. This change now has its effect:

SAM ADAMS WANTS TO GET SOME CHEESE.

SAM ADAMS WANTS TO FIND OUT WHERE SOME CHEESE  
IS.

DOES SAM ADAMS LIKE IRVING BEE?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

Sam doesn't know where any cheese is. The deterioration of his relationship with Wilma has precluded his asking her for this

information. That is, there are conditions on how much affection and trust Sam feels for Wilma in order for him to ask her for information. Those conditions were once met but no longer are.

DOES SAM ADAMS LIKE JOHN BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

SAM ADAMS DIDN'T FIND OUT WHERE SOME CHEESE IS.

SAM ADAMS DOESN'T GET SOME CHEESE.

We've made things rough for poor Sam. There's no one else he thinks he can ask for information. (They might feel quite differently about it.)

DOES SAM ADAMS DOMINATE WILMA CANARY?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*3

SAM ADAMS STRIKES WILMA CANARY.

WILMA CANARY IS NOT AT ALL HEALTHY.

Sam is very upset at Wilma. The only thing that could have prevented him from getting back at her was if he had felt completely dominated by her. The dominance factor is checked at other times, too. In some of those cases, the degree of flexibility is much wider than it is here.

WILMA CANARY WON'T TELL SAM ADAMS WHERE SOME  
BERRIES ARE.

It's not surprising, either.

SAM ADAMS DIDN'T FIND OUT WHERE SOME BERRIES ARE.

SAM ADAMS DOESN'T GET SOME BERRIES.

THE END.

Well, he ran out of things to try. Since his hunger was the "top-level" goal, and it has been resolved one way or another, we're done.

-- DECIDE: DO YOU WANT ANOTHER STORY ABOUT THOSE CHARACTERS?

\*YES

We can continue with this group of characters if we like. The world is just as we left it at the end of the previous story, but we can change the "main character." In the new story, Sam is still hungry and unable to find food, Wilma is still injured, and so on.

THIS IS A STORY ABOUT ...

1: WILMA CANARY      2: SAM ADAMS      3: IRVING BEE

4: JOHN BEAR

\*4

Let's hear about John Bear.

HIS PROBLEM IS THAT HE IS ...

1: HUNGRY      2: TIRED      3: THIRSTY      4: HORNY

\*1

Let's make him hungry and see what he does.

JOHN BEAR IS SOMEWHAT HUNGRY.

JOHN BEAR WANTS TO GET SOME BERRIES.

JOHN BEAR WANTS TO GET NEAR THE BLUEBERRIES.

Of course, he remembers those blueberries that we created back in the beginning.

IN CREATING A VALLEY, WE CAN MAKE UP A NEW ONE  
OR USE AN OLD ONE. DO YOU WANT TO USE ANY OF THESE?

1: \*VALLEY\*0      2: \*VALLEY\*1

-- DECIDE:      \*YES

PLEASE TYPE AN INTEGER BETWEEN 1 AND 2

\*1

The travel agent is at work again.

[ CREATING A NEW VALLEY-PASS BORDER (\*BORDER\*6) ]

[ CREATING A NEW VALLEY-MEADOW BORDER (\*BORDER\*7) ]

[ CREATING A NEW PASS-VALLEY BORDER (\*GATE\*0) ]

[ CREATING A NEW PASS (\*PASS\*0) ]

[ CREATING A NEW MEADOW-VALLEY BORDER (\*BORDER\*8) ]

[ CREATING A NEW CAVE EXIT (\*DOOR\*2) ]

[ CREATING A NEW CAVE ENTRANCE (\*DOOR\*3) ]

JOHN BEAR WALKS FROM A CAVE ENTRANCE TO THE  
BUSH BY GOING THROUGH A PASS THROUGH A VALLEY  
THROUGH A MEADOW.

He comes down from his cave in the mountain.

JOHN BEAR TAKES THE BLUEBERRIES.

JOHN BEAR EATS THE BLUEBERRIES.

THE BLUEBERRIES ARE GONE.

JOHN BEAR IS NOT VERY HUNGRY.

THE END.

There were no obstacles, so the story comes right out.

-- DECIDE: DO YOU WANT ANOTHER STORY ABOUT THOSE CHARACTERS?

\*NO

We're finished.

### 2.3 OTHER EXAMPLES

Here is an example of a mode-3 story generated by TALE-SPIN, where the point of the story was chosen at the beginning. The program produced the story using the Conceptual Dependency representation for meaning, and that output can be fed to another part of the program, discussed in Chapter 12, which expresses it ("translates it") in English. That program produced the English that we saw in the previous section. It is an adequate rendering, but it lacks style, so I present here a translation done by hand, for ease of reading. All the events in the story were produced by the program; only the English is mine.

The Fox and the Crow



Once upon a time, there was a dishonest fox named Henry who lived in a cave, and a vain and trusting crow named Joe who lived in an elm tree. Joe had gotten a piece of cheese and was holding it in his mouth. One day, Henry walked from his cave, across the meadow to the elm tree. He saw Joe Crow and the cheese and became hungry. He decided that he might get the cheese if Joe Crow spoke, so he told Joe that he liked his singing very much and wanted to hear him sing. Joe was very pleased with Henry and began to sing. The cheese fell out of his mouth, down to the ground. Henry picked up the cheese and told Joe Crow that he was stupid. Joe was angry, and didn't trust Henry anymore. Henry returned to his cave.

Here is an example of a story produced by starting in mode 1 and continuing in mode 3 with the same characters. The same mechanism that was used for the trick in the last story will be used here. It is discussed in detail in section 7.4; in section 7.4; it works by having the characters themselves make inferences, both forwards and backwards, connecting the goal of possessing an object to some action on the part of the owner, and then using the information about the owner's personality to find a reason why the owner should perform that action. In the last story, Henry Fox knew that Joe Crow was vain and might therefore be persuaded to sing. In this story, Joe Bear wats Jack Bear to go away (PTRANS). Because they are competitive, Joe knows that Jack is likely to respond to a dare from him. There is a procedure which produces modified conceptualizations given the relationship state. From PTRANS and "competitive," it produces the representation for "run quickly."

Again, the story was written by the program; the English version was written by me.

Once upon a time, there were two bears named Jack and Joe, and a bee named Sam. Jack was very friendly with Sam but very competitive with Joe, who was a dishonest bear. One day, Jack was hungry. He knew that Sam Bee had some honey and that he might be able to persuade Sam to give him some. He walked from his cave, down the mountain trail, across the valley, over the bridge, to the oak tree where Sam Bee lived. He asked Sam for some honey. Sam gave him some. Then Joe Bear walked over to the oak tree and saw Jack Bear holding the honey. He thought that he might get the honey if Jack put it down, so he told him that he didn't think Jack could run very fast. Jack accepted this challenge and decided to run. He put down the honey and ran over the bridge and across the valley. Joe picked up the honey and went home.

In mode 3, TALE-SPIN knows about morals and establishes the social preconditions ahead of time. The only difference in the requirements between this story and the last was the element of competition, not vanity. The moral of that story was "Never trust flatterers." The moral of this one is "Never take a dare."

TALE-SPIN doesn't "understand" the story it's telling, but it's not very different from a program that would. It has a representation for meaning, and the characters can understand each other in a non-trivial way by using inferences to deduce each other's goals. With a parser and a question-answering component, TALE-SPIN could be expanded to understand stories.

## CHAPTER 3

### PLANNING STRUCTURES

#### 3.1 INTRODUCTION

**A**t the heart of TALE-SPIN is a problem solver, a program which implements a new theory of planning. Accordingly, the stories TALE-SPIN produces are essentially accounts of what happened during the course of solving one or more problems. This is consistent with the theory that all stories are about problems (see Chapter 7).

The theory of planning is based on ideas of Schank and Abelson [1,28]. Central to their theory is the idea that there are several different techniques which people use in solving problems. A problem solver based on theorem proving, by contrast, might assume that there was, in essence, one uniform method to use in proving theorems, resolution, for instance. Since we are using several methods, we need a

generic term to describe methods. For this purpose I use the term "planning structures."

Schank and Abelson developed their planning structures to enable understanding of planful behavior, although they claim that a program could use the theory to produce planful behavior [1, page 300]. TALE-SPIN is such a program.

It is necessary to distinguish between using planning structures for understanding and using them for problem solving. Planning structures used in understanding can be thought of primarily as data, patterns against which the behavior is to be matched, while problem-solving planning structures are primarily procedures which are carried out to solve the problem. (This does not mean that the programs which use planning structures need to be implemented in this way.) Since TALE-SPIN is a problem solver, I will refer to the planning structures as procedures.

The planning structures I use from Schank and Abelson's work are delta-acts, packages, and sigma-states. Other planning structures which I introduce here are action modules, reaction modules, relationship states, and personality states.

Delta-acts organize knowledge about the playful behavior associated with a particular goal state. I have implemented the three delta-acts which correspond to the primitive acts PTRANS, MTRANS, and ATRANS. A delta-act is defined as a goal state, a set of particular methods

(called planboxes) for achieving that goal state, and a decision procedure for ordering the planboxes. Each planbox has a set of preconditions which define the order in which the planboxes can be used, if at all. "Unconscious" preconditions are attached to planboxes which would never occur to you to use. If you're trying to get near X, you don't even think about persuading X to come to you when X is an inanimate object. "Uncontrollable" preconditions cannot be made true if they're not already true. (The assumption is that they are sometimes true.) "Plantime" preconditions are the things you worry about when you're making up the plan. You don't worry about runtime preconditions until you're executing the plan. If I'm planning to get a Coke out of the machine upstairs, I worry about having enough money, but I don't worry about walking up the stairs until I'm at the stairs. That the machine actually has some Coke is an uncontrollable runtime precondition: I don't worry about it until I get there, and there's nothing I can do if it is empty when I get there.

There is also a class of "social" preconditions. Although we assume that "Y is near Z" is logically equivalent to "Z is near Y," it may or may not be appropriate to convert the goal of getting Y near Z into the goal of getting Z near Y. For example:

John wanted to see his boss about a raise. He called him on the phone. "Harry, I'd like you to come over here, right now."

That sounds a little bizarre: The relationship state (employee to employer) is being ignored.

Finally, there are "contextual" preconditions, having to do with the pursuit and preservation of goals.

John wanted to fry an egg. He put the pan on the stove, and brought over an egg from the refrigerator.

This sounds fine.

John wanted to fry an egg. He put the pan on the stove, then carried the pan over to the refrigerator to put the egg in it, then brought it back to the stove.

This is worse. Having successfully pursued the goal of getting the pan to the stove, he fails to preserve it. (Sussman's HACKER system [31] addresses this problem in detail.)

Sigma-states correspond to physical needs which have to be satisfied from time to time. As planning structures, they differ from delta-acts in that they do not relate to each other in a goal-subgoal fashion. The need to eat food, for example, arises spontaneously, not as a means of achieving some other goal. Sigma-states are cyclic, arising time after time. Since we get so much practice at solving them, their solution methods can look very much like scripts [28,30], situations which are so familiar that we know exactly what to do and don't have to plan.

The delta-acts and sigma-states relate to each other in a goal/subgoal fashion, but after all the preconditions are taken care of, some specific action must be performed. For example, DELTA-PROX is the delta-act for changing location, and it may invoke DELTA-KNOW to find out some information or DELTA-CONTROL to acquire a means of transportation, but at some point, someone is going to do a PTRANS, an



actual, physical movement. These actions are produced by what I call action modules; they exist as separate entities because there is sometimes additional work (e.g., runtime preconditions) that gets done only during the performance of actions and not during the planning. Such is the case for MTRANS and PTRANS. (DO-MTRANS is described in section 3.12, DO-PTRANS in section 3.13.) The action modules call the assertion mechanism (see the figure in section 1.1) to alter the world state. The simple action-modules (all except DO-MTRANS and DO-PTRANS) simply assert that the appropriate action has occurred; the assertion and inference mechanisms do all the rest.

Reaction modules are actually knowledge-propagation inferences, what things are true now that you've found something out. Joe Bear reacts to the knowledge that Irving Bird thinks he's an idiot, for instance, by feeling somewhat less kindly toward Irving. The reaction modules are described in the chapter on inferences, in section 6.3.

If we're going to say that Irving thinks Joe's an idiot, or that Joe isn't so crazy about Irving either, then we've opened up a whole new set of states, what I call personality and relationship states. They describe what one character thinks about himself and what he thinks about other characters. In this case, we know where Irving thinks Joe is on an "intelligence" scale (personality), and we know how Joe thinks he relates to Irving on an "affection" scale (relationship). These states and scales are described in sections 3.14 and 3.15. In Chapter 4 I discuss the relationship of this approach to problem solving to that

of other systems such as STRIPS.

### 3.2 DELTA-PROX

**D**PROX(X,Y,Z) is the procedure which person X uses to get Y near Z .  
If X is the same as Y, then we're saying that X wants to be near Z.  
Otherwise, Y is some object, possibly a person, which X wants to be located at Z. The goal state is "Y is near Z."

Planbox 0: Does X think that Y is already near Z?

The planboxes (methods) are numbered sequentially from 1, but I use the term "Planbox 0" to indicate a memory check to see whether the goal state is already true. The goal-state check is always done with reference to the person doing the planning. In this case, we check whether X believes that Y is already near Z. If so, DPROX succeeds. If not, we try Planbox 1.

Planbox 1: X tries to move Y to Z  
preconditions: X is self-movable  
                If X is different from Y,  
                    then DPROX (X, X, Y) and DO-GRASP (X, Y)  
                DKNOW (X, where is Z?)  
                DKNOW (X, where is X?)  
                DLINK (X, loc (Z))  
                act: DO-PTTRANS (X, Y, loc (Z))  
postcondition: Is Y really at Z? (DKNOW could have goofed)  
postact: If X is different from Y,  
                then DO-NEG-GRASP (X, Y)

X will try to move Y to Z. If X is not the same as Y, then X must get himself near Y, and then somehow grab on to Y, so that Y will be carried along.

An "unconscious" precondition for Planbox 1 is that X be able to move himself. If that isn't the case, or if the planbox fails, we try Planbox 2.

The next precondition on Planbox 1 is that X find out where Z is. If Z is a fixed physical location (a "map" location), then nothing needs to be done; we assume that everyone knows locations on the map, although we do not assume that everyone knows how to get to every location on the map. If Z is not a map location (Z might be a person, for instance), then we use the delta-act for finding out information, DELTA-KNOW.

The next precondition is that X find out where he himself is. X must know where he is in order to go anywhere. Including this test allows the possibility that X does not know where he is and must therefore find out. Thus, we can now tell stories (solve problems) in which characters are lost.

The next precondition is that X figure out a route to Z. The solution to this problem depends on our theory for representing locations in the physical world, which I discuss in detail in Chapter 11.

Finally, we use the DO-PTRANS action module to effect the actual movement, and if X had picked up Y, he now lets Y go. The DKNOW postcondition allows for the possibility that the DKNOW precondition may have been satisfied unfairly. In Chapter 11 we'll see a story in which Arthur Bear asks George Bird where some honey is. George lies, and Arthur believes him. As far as the first DKNOW was concerned, the precondition was satisfied: Arthur thinks he knows where the honey is. When he gets there, however, he discovers that the honey actually isn't there, so he goes back to do another, hopefully more successful, DKNOW.

Planbox 2: X gets Y to move himself to Z  
preconditions: Y is sentient (one of the characters)  
                  Y is movable  
action: PERSUADE (X, Y, Y PTRANS Y TO Z)

Planbox 2 of DELTA-PROX is for X to persuade Y to move himself to Z. Obviously, Y must be a person and movable. We simply use the PERSUADE package (see section 3.7) to solve the problem.

Planbox 3: X gets someone else, P, to move Y to Z  
precondition: Y is movable  
                  X thinks that P relates to him either  
                          as a friend or as an agent  
action: PERSUADE (X, P, P PTRANS Y to Z)

Planbox 3 calls for X to use a friend or agent to move Y to Z. The notions "friend" and "agent" are discussed in section 3.14. Again, we rely on the PERSUADE package to do the work for us.

Those are the only planboxes currently used in TALE-SPIN. Here are some others that I didn't get around to implementing:

Planbox 4: X gets Y and Z to be in some common location L  
preconditions: Y is movable  
                  Z is movable  
actions: DPROX (X, Y, L)  
          DPROX (X, Z, L)

If Z is a movable object (even a person), then X can choose some new location L and get both Y and Z to wind up at L. For example, if X is the same as Y and Z is also a person, then L might be some place they'd meet for lunch. If Z is an inanimate object, then L could be a place for Z to be delivered.

Planbox 5: Let some system move Y to Z.  
precondition: available system  
action: whatever is appropriate to the system  
          (probably a script)

X can use a natural force, such as gravity, or an agency system, such as the post office, to get Y to Z.

Planbox 6: X gets someone else, P, to move Z to Y.  
precondition: Z is movable  
                  X thinks that P relates to him either  
                          as a friend or as an agent  
action: PERSUADE (X, P, P PTRANS Z to Y)

X can use a friend or agent to move Z to Y.

Planbox 7: X gets Z to move himself to Y  
preconditions: Z must be sentient (one of the characters)  
                  Z must be movable  
          action: PERSUADE (X, Z, Z PTRANS Z TO Y)

X can try to persuade Z to move himself to Y. Obviously, Z must be a person and movable.

### 3.3 DELTA-NEG-PROX

**A**nother entire interpretation of DELTA-PROX is DELTA-NEG-PROX, getting something away from some location. Negative delta-acts can often be replaced by specific positive delta-acts.

John had worked all night at the office. He didn't want to be at the office anymore. He wanted to go home.

But occasionally any place will do. In the TALE-SPIN story where Henry Ant falls into a river, he wants to be anywhere except in that river, so the program selects each of several nearby locations until a successful DELTA-PROX to one of them has been achieved. The "nearby locations" are found by consulting the map of the physical world and selecting locations in the same general area as the river. Geographical information, as described in Chapter 9, is organized in a hierarchy of regions, making this selection process very simple.

### 3.4 TELL AND DELTA-KNOW

**A** belson's DELTA-KNOW concerns the problem of communication. I call that procedure TELL(X,Y,Z) -- X wants Y to know Z -- and it has two simple planboxes.

Planbox 1: X himself tells Z to Y. The precondition for this is DPROX(X,X,Y) -- X has to be physically proximate to Y in order to communicate with him. This is clearly a simplification; communications systems such as telephones would be added in a more elaborate model. If the precondition is achieved, then DO-MTRANS is called.

Planbox 2: X persuades a friend to tell Z to Y.

I define DKNOW(X,Q) as the procedure X uses to find out the answer to question Q. The goal state is "X knows (the answer to) Q." The planboxes under DELTA-KNOW are distinguished by the type of the question.

Planbox 1: Is there a standard reference for finding out the answer? To find out what time it is, for instance, you could go out and make measurements on the sun, but more likely you'll just look at your watch or a nearby clock. These rote responses are represented with scripts. With each standard-reference question, TALE-SPIN associates a script to use.



Planbox 2: Is there someone who is likely to know the answer to the question? For instance, the question "How many moons does Saturn have?" might be easily answered by an astronomer, "How far is Seattle from Portland?" by someone who lives in the Pacific Northwest, and "How much does tomato sauce cost?" by someone fond of spaghetti.

Joe Bear is considered to be an "expert" on bears, caves, berries, and honey: his own species, his home, and his food. This data is part of the initial world knowledge in TALE-SPIN; in sections 11.4 and 11.6, we will see how it is used when characters are created.

Planbox 3: Is this a "general" question, one almost anyone might be able to answer? That question itself is a general question -- almost anyone knows whether a given question is a general question or not. X finds the answer by persuading a friend to tell him the answer. "Where is ...?", "Who has ...?", and "Who knows ...?" are the general questions actually used in DKNOW.

Planbox 4: X persuades an agent to find out the answer and tell him.

### 3.5 DIGRESSION: SPECIAL INFERENCES IN COMMUNICATION

An interesting feature of TELL and DELTA-KNOW is the importance of temporal information. An early version of TALE-SPIN produced a story in which Joe Bear walks up to Irving Bird and asks him where some honey

"will be." The future tense made sense, because Joe was planning to obtain possession of the honey at some future time, not because he was simply curious. Irving must make the appropriate inference in order to understand the question correctly. Consider two examples:

Mary: "John, do you know what time it is?"  
John: "Yes."

As a normal conversation, this seems odd. John has not figured out that Mary wants to know what time it is and wants John to tell her.

The representation we now use for "ask" is literally "ask to tell." That is, Joe doesn't say, "Where is the honey?" but rather "Will you tell me where the honey is?" The future tense is "attached" to the telling. In DO-MTRANS, when one character has spoken to another, we perform a pattern match on the message (see section 11.20) to detect whether it is a request for action, not simply for a yes/no response.

One way of establishing the precondition for Planbox 2 of DELTA-KNOW is to ask someone whether he knows the answer to the question -- DELTA-KNOW as a precondition to DELTA-KNOW -- but if the person says yes, you are not expected to have to ask him for the answer. On the contrary, it is expected that he will understand that the only reason you asked him whether he knew was because you wanted him to tell you the answer.

John was walking down Main Street late one night when a man rushed up to him and asked him where the nearest phone was. "Right inside this bank," said John, pointing to the darkened building next to him.

John's answer is odd because he failed to realize that the man wishes to

use the phone.

This inference saves us from having to be ultra-precise. Only a pathologically literal-minded person (or a computer) would ask, "What time will it be when you tell me what time it will be?"

In TALE-SPIN, the motivational inferences, which can be used by the characters as well as the storyteller, connect events with reasons why those events might occur. Irving can figure out that a reason for Joe to be near some honey is that he can then take it, or that the reason for him to be in his cave might be to go to sleep.

### 3.6 DELTA-CONTROL

**D**CONT(X,Y) is the procedure X uses to acquire Y. I make no distinction here between control and possession, although standard Conceptual Dependency does.

Planbox 0 is the memory check to see whether the goal state is already true. In the case of DELTA-CONTROL, we check memory to see whether X believes that he already owns Y. Suppose "X owns Y" is not in memory but "X is attached to Y" is. Either one will suffice. The latter is an inference from "X grasps Y," using a new state in CD, FIXED, which means either "stationary" (fixed to nothing in particular, like the ground), or "attached" (fixed to something specific, like someone's hand).

Planbox 1 of DELTA-CONTROL tests whether Y is free for the taking, whether X believes that anyone owns Y. If not, we call DPROX(X,X,Y) to get X near Y. Then we need to repeat the tests to see whether X believes that anyone owns Y, since X may have learned that information during the course of the DELTA-PROX. For example:

One day Joe Bear was hungry. He remembered that there was some honey in the beehive in the oak tree. When he got there, he saw Henry Bee....

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving didn't really like Joe a whole lot, so he told him about the honey in the oak tree, but he didn't tell him that it belonged to Henry Bee. When Joe Bear got to the oak tree, he saw Henry Bee....

If there is still no owner, if Y really is free for the taking, then we use the DO-ATRANS action module to get X to take Y.

Planbox 2 of DELTA-CONTROL is used when X wants to trick the owner (who is known by now) into giving Y to X. For example, X can try tricking the owner into causing Y to be near X, so X can grab Y and run. We might view this as part of DELTA-PROX, since an object's location is being changed. However, so much of the context has to do with ownership, and so little with location, that it is impractical to put this into DELTA-PROX, as aesthetically pleasing as that would seem. Context blurs the distinction between one planning structure and another.

The social preconditions on this planbox are that the person using trickery be dishonest. The kind of trick we use depends on the personality of the owner. In the fox-and-crow story in section 2.3, the fox is dishonest and the crow is vain. The fox uses the inference mechanism to determine that if the crow opened his mouth, the cheese would fall near the fox, and that a reason for opening his mouth would be to speak. Some acts, slightly modified, are consistent with certain personalities. A vain character might not be easily persuaded to talk (SPEAK), but he just might be persuaded to sing. He might not be persuaded simply to walk (PTRANS), but perhaps to run fast, or fly very high if he has wings. A person who felt very competitive might be also challenged into running fast or far (PTRANS), or throwing something (PROPEL) a great distance.

The fox figures out that the crow might like the idea of singing, so he compliments the crow's singing and asks the crow to sing for him, which the crow does, dropping the cheese. We then use DO-ATRANS for the fox to grab the cheese, and DELTA-NEG-PROX for the fox to get out of there.

Planbox 3 of DELTA-CONTROL has X persuading the owner to give Y to him.

Planbox 4, the "steal" planbox, calls DELTA-NEG-PROX to get the owner away from Y, DELTA-PROX to get X near Y, and DO-ATRANS for X to take Y. Dishonesty is a social precondition here.

Those are the four planboxes now in DELTA-PROX, but what other planboxes might there be? X could try persuading a friend or agent to get Y and give (sell, trade, ...) it to him. More exotic methods exist. X could get a group of friends to chip in and buy Y for him. He could persuade his rich uncle to leave Y for him in his will (and then, possibly, bump off the rich uncle). He could persuade his rich uncle to leave him enough cash or negotiable goods in his will so that he can buy Y (and then bump off the uncle), and so on. The more detailed the planbox, the more information you need in order to use it. Conversely, the more information you have, the more options you have, the better you are at solving problems. The most detailed planboxes are those that are used in the context of an overwhelming amount of knowledge, especially knowledge from experience. Such detailed planboxes are called scripts by Schank and Abelson [27].

Today, as he walked into the Burger King where he'd had lunch every day for the past month, John wondered what he should do to get a hamburger.

If we now make unflattering assumptions about John's intelligence, no one will blame us much -- everyone expects John to know exactly what to do by now.

### 3.7 THE PERSUADE PACKAGE

**P**ersuasion is organized around an act we want someone to do, not a state. *PERSUADE*(X,Y,Z) is the procedure X uses to persuade Y to do Z.

The *PERSUADE* package is no more than a handy collection of planboxes used by many delta-acts. The main preconditions are social ones: How does X feel about Y? The planboxes are general enough to be used in many contexts. (Those that aren't are left back in the delta-acts.)

Planbox 1 of *PERSUADE* is for X simply to ask Y to do Z (more precisely, to ask Y whether he will do Z). The social preconditions which preclude this are:

1. X is trying to deceive Y, to trick Y into doing Z, in which case, coming out and asking would be inappropriate.
2. X thinks that Y is trying to deceive him.
3. X feels very competitive towards Y.
4. X dislikes Y intensely.
5. X thinks that Y dislikes him.

If none of these conditions are true, then X asks Y (using *TELL*, described in section 3.4). If the request worked, *PERSUADE* succeeds. Otherwise we go on to the next planbox.



In Planbox 2, "Inform Reason," X tries to think of a good reason why Y should do Z. X considers the goals he assumes Y has (eating, for example), and tries to figure out whether Z would lead to one of the goals. If so, X then figures out whether there's something he can do to enable Y to do Z. If there is, then he goes and does it. If Y has caught the hint, he will now do Z, and the "persuasion" is successful.

How is all this "figuring out" and "hint-catching" done? With special kinds of inferences. I describe them in Chapter 6, but suffice it to say that there is a class of inferences which connect states to the acts from which the states result, inferences that connect acts to the states that enable them, inferences that connect acts to reasons why those acts might be done, and so on. Each class is used separately, according to the problem-solving context.

If X couldn't think of a good reason, we try Planbox 3 next. Otherwise, if Y ignored X's effort on his behalf, X gets mad at Y (the "affection" relationship between them changes), and we use Planbox 4.

In Planbox 3, X tries to bargain with Y. Again, X considers Y's goals and the preconditions to one of those goals that he might be able to achieve. (If the goal were eating, the precondition would be to bring Y some food. Not all goals have preconditions which other people can achieve. If Y is tired, his goal is to rest, but there's not much X can do to help him, except maybe to leave him alone.) X then offers to do this thing for Y. If Y agrees, X does his part of the bargain. If Y doesn't agree, or if X fails in his attempt, then X tries to think of

something else to bargain with. If Y doesn't keep his part of the bargain, X gets very mad at Y and gives up the idea of bargaining.

Planbox 4 is called "Threaten," and is the last resort for persuasion. If X likes Y a lot, or feels dominated by him, then he won't threaten, and his attempt at persuasion fails. I currently use only the threat of physical harm, so that if Y outweighs X or is taller than X, X won't threaten. The threat is that X will hit Y with his hand (paw, whatever). If X is very mad at Y, he simply goes ahead and bashes Y. Otherwise, he warns Y first. If es in, then the persuasion succeeds. If not, he hits him, and the persuasion fails.

### 3.8 SIGMA-HUNGER

**T**he procedure for X to satisfy his hunger is simply to obtain some food and to eat it. We first check to see whether X knows about any available food. If not, he tries for any food at all. SIGMA-HUNGER calls DCONT(X,food) to get him the food, and DO-INGEST to have him eat it.

Like all states in CD, hunger is represented on a scale from -10 (starving) to +10 (stuffed), and the goal of SIGMA-HUNGER is to go from a negative value to a non-negative value. There are ways of losing one's appetite without eating (a bug in the salad, for instance), and they might even make for interesting stories, but they're not implemented.

The resultant inferences from INGEST-FOOD are that the food no longer exists and that X is less hungry.

### 3.9 SIGMA-THIRST

**S**IGMA-THIRST calls DPROX(X,water) and, if that succeeds, DO-INGEST(water).

In the world which TALE-SPIN currently simulates, water is found in rivers and is owned by no one. Furthermore, while Joe Bear might be able to eat all the berries on a bush, he certainly can't drink all the water in a river: the water isn't "all gone" afterwards. The resultant inference from INGEST-WATER, then, is only that X is less thirsty.

### 3.10 SIGMA-REST

**E**ach character in the story has a home where, presumably, he can go in order to sleep, so SIGMA-REST calls DELTA-PROX to get the character home. Sleeping itself is represented as a script. That is, it requires no problem solving, although one might include episodes of insomnia, indigestion, mosquitoes, nagging backaches, maddening itches, and other various ailments to prevent or interrupt sleep.

The inference from \$SLEEP (the sleep script) is an increase in the sleeper's state of REST.

### 3.11 SIGMA-SEX

**T**he character chooses one of his very good friends. The reader gets to decide the character's preferences as to same-species/different-species, same-sex/different-sex. The character simply tries to persuade the other person to "fool around," as the generator says. The possibilities for refinement are numerous.

The inferences from \$SEX are a decrease in the need for sex and an increase in the need for rest.

### 3.12 DO-MTRANS

**I**f X is communicating information to himself, that is, remembering something, no special work needs to be done. If X is telling Y some "new" information, we will have to create the new data. That is, part of the process of making up a story "on the fly" is that you can introduce new objects and characters at will.

Suppose he tells him where the honey is. Is that enough? Probably not. If he knows who owns the honey, and whether the owner is at home, he might also say these things. The context in which the information is

asked determines the appropriate response.

This also opens up possibilities for lies. Irving Bird can tell Joe Bear about some non-existent honey, or he can tell him about some real honey but deliberately not tell him that the honey belongs to Henry Bee, or he can tell him that the honey belongs to *someone else*, or he can lie about the location, and so on. The example in Chapter 2 used just such a trick. The decision whether to lie depends on the degree of deceit in the relationship.

It is in DO-MTRANS that requests are handled. That is, the representation of the message enables us to detect when one character is simply passing along information and when he's asking someone to do something. When X considers a request from Y, he checks whether their relationship dictates the response, regardless of what it is that Y has asked X to do. So if X dislikes Y, or distrusts him, or feels very dominant over him, he refuses. On the other hand, if he likes him very much, or feels indebted to him, or feels very submissive towards him, he agrees. Otherwise, he considers the effects of doing what Y has asked, by computing the "automatic" inferences (but not asserting them as true), and considering his reactions to them. If the reaction is strong, X will agree or refuse, as appropriate. Otherwise, if the relationship doesn't say whether to do it or not, and he has no reaction either way, we check X's personality and see how kind he is, and agree or refuse on that basis.

Promises are also handled in DO-MTRANS. They arise in bargaining situations, where Y asks X if he will do A if Y does B. If X is being very deceitful, he will say yes but plan to renege and, in fact, insult Y after Y has done his part of the bargain. Otherwise, if X decides that he likes the agreement, he plans to do A as soon as Y does B, and he tells Y that he agrees. If he doesn't like the deal, he refuses.

### 3.13 DO-PTRANS

The special handling here has to do with means of transportation. Characters who walk stay on the ground, so when Joe Bear goes to Irving Bird, he winds up at the ground by the tree, not up in the nest with Irving. The notion "X is near Y" is extended to include such cases. See the example in section 11.16.

### 3.14 RELATIONSHIP STATES

Agreeing on the general idea of social preconditions -- that X must like Y before asking him to do a favor, for example -- is easy, but agreeing on the specific choice of relationships is more difficult. The relationships we use are competition, dominance, familiarity, affection, trust, deceit, and indebtedness. The question is: Why these and not others? The aesthetic goal is to choose the smallest "linearly independent" set that explains all the things we're interested in

explaining, but finding the primitives for social relationships is not easy. The choices of competition, dominance, and familiarity are based on research by Myron Wish [34,35]. The others have been added when I needed them to write a story and wasn't satisfied that they were combinations of the relationships already in use. While the program certainly does not permit all possible combinations of scales and values, it does use each one of them in the model; that is their real justification for existence.

Consider what it means for a computer program to understand the idea of affection. It is not that it will now print out "John loves Mary," but rather that it will use that information in modeling (predicting) John's behavior, so that if Mary asks him to do something, for example, the fact that he loves her enters into his decision whether to comply with her request.

In his recent book, Weizenbaum made the following statement [33, page 200]:

It may be possible, following Schank's procedures, to construct a conceptual structure that corresponds to the meaning of the sentence, "Will you come to dinner with me this evening?" But it is hard to see ... how Schank-like schemes could possibly understand that same sentence to mean a shy young man's desperate longing for love. Even if a computer could then simulate feelings of desperation and of love, is the computer capable of being desperate and of loving? Can the computer then understand desperation and love? To the extent that those are legitimate questions at all, and that is a very limited extent indeed, the answer is 'no.'

TALE-SPIN does simulate feelings of love in the story's characters.

Does the program feel love? Does the time-sharing system on which it



runs feel love? Does the hardware feel love? Of course not; how absurd. As far as being able to get the computer to understand love, we are restricted to describing love in practical terms ("Love is when ....") because computers can only sympathize, never empathize. But that's sufficient for any reasonable definition of "understanding." By Weizenbaum's logic, computers can't "understand" even the number zero, much less the notion of love, nor can people who are blind from birth "understand" vision.

Returning to details of the program, the representation for relationships always includes a person who believes that the relationship exists. Thus, "John loves Mary" is actually shorthand for "John believes that he loves Mary." There are two reasons to do this. First, I'm not sure it means anything -- in the technical sense -- to say that John loves Mary but he doesn't believe that he does. If it does, it's very subtle. Second, the long version uses the same form of representation as "Mary believes that John loves Mary," "Fred believes that Mary loves John," and so on.

It's also very important to us to be able to distinguish "John loves Mary" from "Mary loves John," so that "John loves Mary" does not contradict "Mary does not love John." The asymmetry of interpersonal relationships is a great source of social dynamics. From the storyteller's point of view, such things tend to keep the story lively, giving rise to new goals all the time.

Each of the delta-acts was a procedure for achieving a certain state. The relationship states are just that, states, but are there also achievement procedures -- call them rho-states -- associated with them? Is there what we might call RHO-AFFECTION( $X, Y, Z, V$ ), a procedure which  $X$  uses to get  $Y$  to like  $Z$  by amount  $V$ ? How would rho-states differ from other planning structures? By design, the rho-states would be independent of each other. That is, RHO-AFFECTION would not "call" RHO-COMPETITION, which makes them very different from the delta-acts. We said earlier that sigma-states were never subgoals -- you don't satisfy hunger as a means of achieving some other goal. Rho-states, it would seem, are always subgoals. If John works hard to get Mary to trust him, he must do so for a reason. But perhaps the most important distinction of the rho-states is that there is much less certainty about them. While there are guaranteed procedures for John to get from New York to London, there are no such guaranteed procedures for John to get Mary to like him, which says no more than that the "physics" of the real world is simpler (or better understood) than the "physexics" of the mind.

The only rho-state used in TALE-SPIN at present is RHO-AFFECTION. It is used as part of the trick planbox of DELTA-CONTROL, as we saw in the fox-and-crow fable. The fox wants to crow to like him, so that when he asks the crow to do something, the crow will oblige. To do this, he compliments the crow, saying that the crow's singing pleases him very much. The increase in affection is part of the crow's reaction to this information.

### 3.15 PERSONALITY STATES

**K**ind, vain, honest, intelligent: these describe individuals, not how individuals relate to other individuals. In considering a request, a character decides whether to oblige based on his relationship with the person making the request, or, failing a decision there, on his reaction to the consequences of obliging, or, failing a decision there, on how kind he is. A compliment produces a much stronger reaction in a vain person than in someone not so vain. Dishonesty is a precondition to the various "trick" planboxes. When someone falls for a trick, he is called stupid, and he reacts very strongly, very negatively, to being called stupid.

I have not implemented planning structures for changing personalities. While such things may exist, they operate over such long periods of time that I doubt people use them in plans, as subgoals.

Mary needed a new dress and wanted to get some cash from her husband. "How can I do that?" she wondered. "Oh sure, I'll just make him dishonest so he'll rob a bank. Gee, I better make him smart enough not to get caught. Okay, that'll work. John?"

Not exactly believable. She might be able to coerce him into robbing a bank, under threat of death, for instance, but then the reason he robbed the bank would be explained by the goal hierarchy: Preserving HEALTH, in particular, avoiding HEALTH -10, is worth almost anything. A real change in personality can be a goal -- education, rehabilitation, adjustment -- and when it is achieved, different planning techniques will be available. In this respect, they differ from the sigma-states,

which primarily affect the scheduling of plans, not the choice of plans.

## CHAPTER 4

### THEORY OF PLANS

This section describes the nature of the plans we use, and distinguishes our problem-solving techniques from those in GPS (Newell and Simon [18]), STRIPS and its successors (Fikes, Hart, Nilsson, and Sacerdoti [7,8,9,23]), and BUILD (Fahlman [6]).

GPS views problem-solving as the task of reducing the difference between the current state of the world and the desired state of the world by applying successive transformations which are known to be useful. This technique, called means-end analysis, is also used in STRIPS, a problem-solver at SRI, where the world model consists of a robot, some boxes, and some rooms. The operations which the robot can perform, transforming the world model from one state to the next, are things like MOVE-ROBOT and PUSH-BOX. To determine whether something is true within a given world model, STRIPS uses resolution theorem-proving. The states are represented in predicate calculus. ATR(Z) means "the

robot is at location Z."  $AT(X,Y)$  means "object X is at location Y." Rules in the world model are also expressed in predicate calculus. "An object can exist in only one location" is expressed as "for all X, Y, and Z, if X is at location Y and Y isn't location Z, then X isn't at location Z." Their procedures consist of preconditions and effects. Preconditions must be provable true before the procedure can be applied. The effects are expressed as things which are now true and should be added to the world model and things which are now false and should be deleted from the world model. For example, the preconditions for pushing object K from location M to location N are  $ATR(M)$  and  $AT(K,M)$ . The newly true things are  $ATR(N)$  and  $AT(K,N)$ . The newly false things are  $ATR(M)$  and  $AT(K,M)$ . STRIPS produces a list of operators (procedures) which, when applied to the initial world state one after another, will result in the desired (goal) state.

By observing patterns in things that become true and false, STRIPS can effectively turn constants into variables and produce general plans called MACROPS that still achieve the desire goals. A program called PLANEX monitors the execution of plans, and when something goes wrong, tries to salvage as much of the old plan as possible instead of solving the entire problem again from scratch. Finally, in ABSTRIPS, problems are organized into levels called abstraction spaces, where unimportant details are ignored. The shape of chess pieces, for example, is unimportant to the game of chess, although it is potentially important to a robot-hand that has to move the pieces. Searches for solutions are confined to single levels whenever possible. Preconditions are ranked

according to "criticality," indicating the level at which the precondition need be considered. Problem-solving starts at the highest, most abstract level, gradually working its way down to the bottom level.

BUILD is a blocks-world program written by Scott Fahlman at MIT. It constructs plans for assembling various configurations of toy blocks. It knows about gravity, friction, equilibrium, and so forth, with respect to blocks, and it uses "heuristically-guided search and controlled backup" (via CONNIVER) to construct plans.

There are many similarities between these systems and TALE-SPIN's problem-solving component. There have to be. How can you argue against the idea that there are goals and methods and preconditions and side effects?

But there are differences, too. One of the important differences between our view of plans and theirs is contained in our notion of a problem domain. A problem domain contains: (1) a set of representational primitives, (2) a set of problems which are expressed in terms of those primitives, and (3) a set of problem-solving procedures. The procedures produce solutions which may involve other domains (subdomains); hence the term "primitive" is merely relative. Since the problems are defined by the primitives, the number of procedures is proportional to the number of primitives.



In GPS and basic STRIPS, all problems were lumped into one problem domain. Everything looks the same. ABSTRIPS approached the idea of separate problem domains, but all the domains used the same set of primitives. High-level information should not be written in low-level terms. Fikes, Hart, and Nilsson [8, page 427] mention that "... the level of detail in a MACROP description does not match the level of planning at which the MACROP is to be used." BUILD, like all the blocks-world programs, suffered from too narrow a world model. Fahlman says:

The world of blocks is ideal for this type of study [robot problem-solving] because it provides difficult and interesting problems, but nevertheless is simple and self-contained. Since the robot needs to know only a few concepts about gravity, support and friction, it is possible in such a world to study the organization of planning programs without facing the difficulties of collecting, maintaining, and effectively using a huge body of real-world knowledge. Games such as chess share this closure property, but are farther removed from the type of useful real-world activity which we would like the robot eventually to perform.

I disagree. While it may be intuitively appealing to believe that it is better to have one set of primitives to represent absolutely everything, it really isn't a simplifying assumption. On the contrary, it goes against the organization of knowledge, the making of distinctions, which is such an important characteristic of human intelligence.

Let me give an example. Here are six sentences which are superficially very similar.

1. John is the owner of the house.
2. Hartford is the capital of Connecticut.

3. Four is the square root of sixteen.
4. G is the dominant of C.
5. Mice is the plural of mouse.
6. Deuterium is the second possible isotope of hydrogen.

I'm assuming that those are all simple facts, not metaphors like "Kissinger is the Disraeli of the 1970's." While those six sentences all look similar in English, they come from very different domains. I claim that people don't think of them in the same way, and therefore, that computer representations for them don't need to look alike. Certainly you can represent them all with semantic triples ("X is the Y of Z"), but that doesn't help, it hurts. It lumps them all together and makes them all look the same, which is very tempting with a small body of knowledge, but which becomes unmanageable with a large body of knowledge. Memory retrieval techniques vary from one domain to the next -- one cannot imagine that in searching for the answer to the question, "What is the capital of Connecticut?" we come across the idea that four is the square root of sixteen. Problem-solving techniques also vary. One cannot imagine that we figure out the square roots of numbers in the same way in which we figure out possible isotopes of the elements.

My claim mentioned "people," and that's another aspect in which this theory of planning differs from STRIPS, perhaps the most important one. We want this to be a theory of how people think about problems, and that's a severe constraint, but it enables us to make some simplifications, too. For instance, people don't worry about all the

preconditions in their plans ahead of time, so this theory makes a distinction between plantime and runtime preconditions. We're not modeling robots on Mars; we can afford to make mistakes, to tolerate some problems along the way, because coping with them -- not planning but re-planning -- is also an interesting human characteristic that we want to put in our model. The MACROPS had lots of low-level preconditions. If we applied MACROPS to our simulated humans, and had them worry about all those nitty-gritty details all the time, we might succeed in modeling neurotics, but that's not our aim.

The STRIPS programs also learn to solve problems. A MACROP is a generalized plan which is constructed by the system. Our theory of planning doesn't touch the learning problem. Joe Bear doesn't learn how to find honey; he already knows. He uses the delta-acts, and they don't vary. The delta-acts were built by hand. They say, for instance, "If you want to get somewhere, here are the six things you can try." We do not deduce those six things. How do we know we have the right ones? We don't yet. We put in the ones we think are reasonable and see how well they survive. What if a character picks one that fails? If he should have known better than to pick that one, then the preconditions are wrong. If it looked okay but failed for extraneous reasons (another character lied to him, for instance), he has to re-plan, but that method wasn't wrong per se.

It's always nice to say that a new theory makes old problems easier to solve, and there are certainly some old problems that look downright trivial in our theory. The classic "Monkey and Bananas" problem reduces to DELTA-PROX, involving a few PTRANSES and GRASPs and maybe an INGEST at the end. But there are also problems which it's harder to solve with this theory, like "Tower of Hanoi" or, to use Fahlman's examples, equilibrium of structures built of toy blocks. "Monkey and Bananas" is easy because that's in the common, everyday problem domain. By comparison, the equilibrium problem, as Fahlman does it, is in a very strange domain, with a whole new set of primitives (those of kinetics) and techniques (solving equations). We haven't begun to model physics in that detail yet. There are two points to be made about the relationship of BUILD to our theory. First, if we thought that that's how people solved equilibrium problems, we'd add that -- integrate that -- in our system, but we'd view it as a separate domain, not one that was typical of the rest of the world of problems. THE BLOCKS WORLD IS NOT A METAPHOR FOR THE REAL WORLD. Second, while I believe that there are people who have to solve equilibrium problems by resorting to equations, I also think it's obvious that other techniques exist, and it might be just as interesting to model how first-graders know that one set of blocks is in a state of equilibrium and another set isn't.

Another notion important to our theory is that of the canned solution. A canned solution to a problem requires no "cognitive" problem-solving in its problem domain or in any other. They are stopping points, beyond which it is assumed that all problems have been

solved. Solutions to many problems are all (or mostly) canned; no one thinks much about getting from room to room. My plan for leaving a room may be simply to walk out. Walking is a canned solution. It comes from a domain where the primitives are related to coordinating muscles and inner-ear balance mechanisms. But I became very good at doing that long ago, so I never worry about it any more; I just do it, and it works. So we don't have to solve problems "all the way down" the hierarchy of domains: "Now I have to move my right leg. Now I have to leave it where it is and move my left leg without falling over...." Those domains are available if we need them, however. Need typically arises in the event of failure. For example, you normally don't think about balance when you hear that John walked across the room, but it can easily come to mind when you hear that John tripped while walking across the room.

Problem domains may or may not relate to one another in a hierarchical fashion. It is common, for example, for one domain to provide "instrumental" solutions to the problems of a higher domain, as in the walking example. But the everyday level does not relate to the domain which includes the problem of analyzing the formal structure of a particular Bach fugue. That musical domain certainly has a full set of primitives and techniques, but there is very little in common with the everyday level of problems. It is not instrumental to any everyday primitive, nor does it require any.

It's hard to imagine what stories you could tell about a robot who pushes boxes around from room to room, so for the particular purposes of this thesis, I am forced to use a variety of domains. One of the more interesting ones from the viewpoint of problem solving is that of interpersonal relationships. In short, there are no algorithms, only heuristics, no proofs, only educated guesses. There's no way the fox can be certain that the crow will like him if he compliments the crow's singing. All he knows is that, based on his beliefs, it's likely.

To summarize, this theory of planning stems from a desire to model humans, and it recognizes and makes use of the fact that there are separate problem domains, areas of knowledge with their own representations and problem-solving procedures. Some preconditions of goals are considered during planning; others are postponed until the plan is carried out. Some problems are assumed to be simple enough not to require any planning at all. This theory does not explain how plans are learned; we assume that the planners use a pre-defined set of general problem-solving techniques.

## CHAPTER 5

### THEORY OF GOALS

#### 5.1 GOAL HIERARCHY

A theory of goals is a theory of the control structure of a problem-solving system. That structure can be divided into two parts, planning and executing. Planning is a purely mental activity in which we decide how to achieve a goal. The execution of a plan describes the steps that were actually taken. Substituting one goal for another and changing the order in which goals are to be achieved, for example, are part of the execution process, not the planning process. Plans are often incomplete, unspecific, and incorrect, since we have imperfect knowledge of the world. Accordingly, much of our activity can be viewed as compensating for our lack of knowledge.



(One can plan to plan. "John decided that he would meet with his campaign workers after the Illinois primary." This is not anomalous, just complicated.)

While deciding how to achieve a goal, people make some calculations. How much is it worth? How much does it cost? When must it be done? I make a distinction between a cost calculus and a time calculus. Cost calculus tells us how well a particular solution matches its constraints. Time calculus says which goals are active and which are dormant, and schedules the active ones.

I assume that goals are states, and that acts which people claim as goals either result in states which were the real goals or are idiosyncratic events which cause some state of happiness.

Achievement and preservation are the two processes which we apply to goals. The planning structures TALE-SPIN uses to achieve goals are discussed in Chapter 3.

It may seem odd, at first, to call preservation a process. Perhaps maintenance is a better word, more easily suggesting activity. Some states require no maintenance. If you own a copy of TV Guide, you will continue to own it until you throw it out. Some states cannot be maintained past a certain point, no matter what. We are doomed to be hungry several times a day as long as we live. But most states are somewhere in between -- with appropriate effort, they can be maintained arbitrarily long.

For example, each of the body's systems might be characterized, on a scale from -10 to +10, by its needs and by its ability to function. Hunger, for example, is the state of need of the digestive system. Deafness is a functional disability of the hearing system. While functional ability is a state, we rarely view it as a goal state. Some systems repair themselves, some require the attention of medical specialists, and some are irreparable. By and large, they're outside the everyday domain. Not even all the need-states concern us. The skin, for example, needs exposure to air, but not counting the last time you saw the movie "Goldfinger," when's the last time you thought about that need? The important physical states are those that commonly come to our attention: hunger, thirst, sex, rest, cleanliness. These needs appear on their own. Each of them can also be preserved for some length of time. How large a meal you choose to eat, for example, may depend on how long you need to preserve your state of non-hunger. The other physical states, lumped together under the single term "health," are usually preserved, and invoke the achievement process when they are not. So in addition to talking about PI-HEALTH (staying healthy), we can also talk about SIGMA-HEALTH (becoming healthy).

What is the nature of the process of achieving a goal? You start with a goal and a set of requirements related to that goal, the constraints. You use your knowledge of plans to come up with a proposed solution, a way of achieving your goal. The solution has its own set of requirements, the facts. The process of goal achievement is that of matching the constraints of the goal with the facts of the solution.

For example, a constraint on buying a pair of shoes might be that they cost less than \$30.00. A given pair of shoes may, in fact, cost \$24.95.

There are actually two classes of constraints and facts, those pertaining to the initial achievement of the goal, and those pertaining to the maintenance or preservation of the goal.

Goal constraints can be thought of as tolerance ranges, permitting some flexibility in the solution. There are three kinds of constraints.

1. Object-specific or local constraints. These constraints relate only to the particular goal state and have no side-effects. If you want a pair of shoes, an object-specific constraint is that the shoes fit.
2. Cost constraints. How does this goal relate to states which we wish to preserve? Cost is expressed in terms of each active pi-state. Common pi-states are effort and money.
3. Time constraints. How much time can we afford to spend on this goal? What times are available to us for achieving this goal?

Facts can be thought of as constants. The cost- and time-facts of a solution are direct analogues of the cost and time constraints. Of the enormous number of object-specific facts, you only consider two kinds.

1. Facts corresponding to object-specific constraints -- You need to know whether the shoes fit, but you may not care whether the laces are flat or round.
2. Subgoals -- These are achieved in the same way as goals, except that there is a new set of constraints for the subgoal, usually an enlarged or more specific version of the old set. For example, suppose PI-MONEY dictates that you may spend no more than \$30.00 to buy a pair of shoes, and suppose that you need to take the bus to the shoe store downtown. You know that you may not spend more than \$30.00 minus the round-trip bus fare when you buy the shoes, or alternatively, that you may not spend more than \$30.00 minus the cost of the shoes when you take the bus. (With a very complicated DELTA-PROX, the alternative would be more reasonable.)

The cost calculus and time calculus exist to handle side-effects. They decide whether one goal blocks another, or whether you can kill two birds with one stone.

TALE-SPIN's goal mechanisms are not yet this complicated. Each character has his own goal structure. When a new goal is being considered, we check to see whether it is already a goal, in which case the new goal fails.

If a goal was added to the structure, but the plan for achieving that goal failed, the goal is not removed from the structure. If it were, that same plan might be used again, despite the fact that it won't work.

Finally, characters can influence each other's goal structures. The entire *PERSUADE* package is designed to do this, to have one character give goals to another character. The trick in *DELTA-CONTROL* alters the relationship of affection between two characters.

## 5.2 COST CALCULUS

What do you do when a plan fails? You consider alternatives. But what if there are several? For example, suppose we had a very simple set of persuasion rules: ask, then offer money. John wants a ticket to a football game and he knows that two of his friends, Tom and Bill, each have a ticket. Suppose John asks Tom for his ticket and Tom says no. What should John do next, offer Tom some money or go ask Bill? If he offers Tom \$5 and he still says no, should he offer \$10 or ask Bill? The answer could depend on where Tom and Bill are. If they're standing next to each other, then it might be reasonable to ask one, then the other, then offer the first one some money, then the other, and so on. On the other hand, if they're miles apart, it could be expensive to bounce back and forth.

In general, you choose the cheapest move which is available to you.

Cost is measured, as we said, in terms of the pi-states, those things like health and money which we wish to preserve. Time is unusual because you spend it even while you're planning -- as in games of five-minute chess -- and because it cannot be recovered once spent -- you may get your money back if you return your new shoes, but you're not going to get the hours back.

What kind of mechanism do we need to represent this? Something which distinguishes not only between one pi-state and another, but also between different references to the same pi-state. To use the example from the previous section, the original pi-money specifies \$30, and that is to be split into two pi-money states, one for the transportation and one for the shoes. Since these have to be accomplished one after the other, we can use a stack, using the heuristic that you never add a state to the stack which costs more than is specified by the current top of the stack. In other words, you know you have \$30 to spend, so you can't afford to spend more than that on the transportation. Suppose that costs you \$5. Now you know that you can't afford more than \$25 for the shoes. If you think that isn't enough for the shoes you want, you can try looking for cheaper transportation, or you might consider allotting more money to the whole project.

The mechanism for a set (or list) of pi-states, then, would be a list of stacks. This is the cost structure.

The unplanned acquisition of knowledge ("noticing") requires that we add another twist. During the planning phase, we need to remember (with pointers into the stacks, perhaps) what the cost structure was. Why? Consider the following story.

Joe Bear was hungry, so he decided to ask his friend Irving Bird where some honey was. But when he got to Joe's tree, he noticed a beehive sitting in the tree.

We want Joe Bear to abandon his goal of asking Irving a question, including all the costs associated with that. Unless we remember what the cost structure was, he can't. Of course, not all those costs can be recovered, such as the cost of the trip to Irving's tree.

Goal specification is the process of deciding which particular goal you want, given a set of possibilities. Joe Bear decides he wants "some food." That may be specified to "some honey," which may in turn be specified to "the honey in the oak tree." The noticing mechanism must check not only that some goal is matched exactly, but also that a higher, less specific goal is also matched.

Joe Bear was hungry, so he decided to ask his friend Irving Bird where some berries were. But when he got to Joe's tree, he noticed a beehive sitting in the tree.

We want Joe to abandon even his goal of finding berries.

TALE-SPIN's current goal mechanisms aren't yet this sophisticated, but it's clear that they could be improved this way. In the REQUEST procedure, for example, when one character is deciding whether to do a favor for someone else, he could view the worth of doing that favor by comparing the resultant states with his own goals. Everyone has the



goal of being happy, so if one of the consequences of doing the favor is that he'll become happier, then he'll be inclined to do it. But more generally, if one of the consequences of doing the favor matches any of his current goals, then he should be inclined to do the favor. Since the current goals change all the time, he may respond differently from time to time when asked to do the same favor, which seems to correspond nicely to actual human behavior.

An addition to the goal mechanism would be to know when goals that have been dubbed "unachievable" should lose that status. They presently stay that way forever, in order to prevent further attempts at solution. So if Joe Bear gives up trying to find some honey, and someone tells him where some honey is, he should again try to get it. To implement this, we need to remember why the goal failed. When new input contradicts that reason (see Chapter 6), the goal can be re-activated.

Two final remarks. First, this model may seem complicated, but it's only an approximation to what people really do. The problem domain here is called Economics, and if we haven't solved all the problems there yet, no one will be surprised.

Second, this isn't the stuff that stories are made of. It may be coherent, but it isn't normally interesting. It can be interesting when the stakes are very high, such as when HEALTH goes to -10 (death), or when MONEY goes to +10 (millionaire).

## CHAPTER 6

### INFERENCES AND MEMORY

#### 6.1 KINDS OF INFERENCES

**I**nferences are the links in a causal chain. We normally think of inferences as consequences, and that is the sense in which the term is used in most of this thesis. But our understanding of the term is greatly expanded by the work of Rieger in his thesis on memory [21]. He described sixteen kinds of inferences used in understanding text, and wrote a program to handle examples of all sixteen types. Here are some examples [25]; the inferences are in upper case.

1. Specification Inferences.  
John picked up a rock.  
He hit Bill.  
JOHN HIT BILL WITH THE ROCK.
2. Causative Inferences.  
John hit Mary with a rock.  
JOHN WAS PROBABLY MAD AT MARY.
3. Resultative Inferences.  
Mary gave John a car.  
JOHN HAS THE CAR.

4. Motivational Inferences.  
John hit Mary.  
JOHN PROBABLY WANTED MARY TO BE HURT.
5. Enablement Inferences.  
Pete went to Europe.  
WHERE DID HE GET THE MONEY?
6. Function Inferences.  
John wants the book.  
JOHN PROBABLY WANTS TO READ IT.
7. Enablement-Prediction Inferences.  
Dick looked in his cook book to find out how to make a roux.  
DICK WILL NOW BEGIN TO MAKE A ROUX.
8. Missing Enablement Inferences.  
Mary couldn't see the horses finish.  
She cursed the man in front of her.  
THE MAN BLOCKED HER VISION.
9. Intervention Inferences.  
The baby ran into the street.  
Mary ran after him.  
MARY WANTS TO PREVENT THE BABY FROM GETTING HURT.
10. Action Prediction Inferences.  
John wanted some nails.  
HE WENT TO THE HARDWARE STORE.
11. Knowledge-Propagation Inferences.  
Pete told Bill that Mary hit John with a bat.  
BILL KNEW THAT JOHN HAD BEEN HURT.
12. Normative Inferences.  
John saw Mary at the beach Tuesday morning.  
WHY WASN'T SHE AT WORK?
13. State Duration Inferences.  
John handed a book to Mary yesterday.  
Is Mary still holding it?  
PROBABLY NOT.
14. Feature Inferences.  
Andy's diaper is wet.  
ANDY IS PROBABLY A BABY.
15. Situation Inferences.  
Mary is going to a masquerade.  
SHE WILL PROBABLY WEAR A COSTUME.

16. Utterance-Intent Inferences.  
Mary couldn't jump the fence.  
WHY DID SHE WANT TO?

The role of inference-making in a simulator is different than in a program which understands input text. TALE-SPIN uses inferences primarily to define the consequences of actions that come from plans. The delta-act DPROX calls the action module DO-PTRANS which asserts that a PTRANS-act takes place. From "Joe Bear walked over to the maple tree" we can infer "Joe Bear is near the maple tree." Asserting an act means adding it to the memory/history of the simulation along with all its consequences, so the resulting change of location also becomes part of the simulation. The characters themselves use the same mechanisms as the storyteller to make those and other kinds of inferences. When Joe Bear tries to think up a good reason why Henry Bee should give him some honey, he needs to work "backwards" from Henry's goals, seeing whether Henry's giving Joe some honey is a causal predecessor of one of the goals. For example, one of Henry's goals is to satisfy his hunger, which he can do by "eating" a flower. In order to eat a flower, you have to "control" a flower, which results from someone (possibly you yourself) ATRANSing the flower to you. We test whether what Joe is trying to PERSUADE Henry to do matches ATRANSing a flower. It doesn't. (Joe is trying to PERSUADE Henry to ATRANS the honey to him.) We then consider that in order to ATRANS a flower, you have to be near the flower, which results from someone PTRANSing you to the flower. Does this match? No. We repeat this process a few times, trying to

construct a short inference chain which connects what Joe is trying to persuade Henry to do and one of Henry's goals. This one failed, but suppose Joe later decided to steal the honey from Henry, which requires that he get Henry away from the hive. The same procedure is used, and we get to the idea of someone PTRANSing himself to a flower again as we did before, but we notice that this does match what we are trying to persuade Henry to do, to PTRANS himself from the beehive to the flower. Joe then considers the precondition for Henry's PTRANSing himself to the flower, namely, that Henry has to know where the flower is, so Joe knows that he has to tell him.

In another example, when Henry Fox figures out, using the enablement of inferences, that he can get the cheese Tom Crow is holding if Tom drops it, he needs to figure out motivational inferences: What could cause Tom to drop the cheese?

Some inferences are "context-free"; that is, they are always true. "John goes to New York" always implies "John is in New York" no matter what. But most inferences rely on memory, and some even require information which, if not in memory, must be asked of the reader.

One way to view the work that is being done on plans and delta-acts and scripts and so on is as a search for causalities, for causal contexts, so that the process of making inferences is more directed, more specific. Inferences are the missing pieces, those parts of the causal context which were not stated because anyone who has figured out what the context is also knows what those missing pieces are.

Working with inferences is working with data: you start with input data, and you get back more data -- the inferences, whether one, two, or sixteen. Working with causal contexts can be like that, and a program which understands plans probably would be like that, but TALE-SPIN is generating plans, and in a basically process-oriented manner, not data-oriented. The point is that the entire organization of the causal context is radically different in a plan-generator than in a plan-understander. While they may share a lot of knowledge, the representation of that knowledge will differ, so I can't always point to a particular spot in the code and say, for instance, "This is where action prediction inferences are made." They're made in many places, in every planbox.

Consequences of events are made explicitly, however, as part of the basic control structure of the system. Consequences include several of the sixteen classes defined by Rieger -- TALE-SPIN has no need to separate them. Three other classes are made explicitly: reactions, which are knowledge-propagation inferences which affect primarily the relationships between characters (listed in section 6.3); enablement inferences, used by the characters (not the storyteller) when they themselves are planning; and motivational inferences, used by the characters in figuring out other characters' plans.

## 6.2 CONSEQUENCES

**F**

or those who want to see exactly which consequences are computed from an event, I list them here. The next section lists the reaction-inferences. Both sections are fairly low-level, and you may want to skip to section 6.4, "Organization of memory."

Temporal relations vary from one type of consequence to the next. Most inferences follow immediately (by 1 time unit). Others, particularly the mental acts and states, have more complicated temporal relations. The change of location following a PTRANS has a time based on the distance traveled.

In the table below, the events appear on the left, their consequences on the right. I use a plus sign (+) after an inference to indicate that the inference is made known to everyone who is "close" to the scene of the event (see section 11.16, "How close is close?"), including the participants in the actual event.

### ATRANS

John gives Mary the book.	Mary has the book. + John doesn't have the book. + Mary knows that John gave her the book.
Mary takes the book from Bill.	Bill doesn't have the book. +

### FIXED

The book is not stationary.	The book falls. + (Gravity propels the book.)
The book is not attached	The book falls to the ground



to John.

near John. + (Memory is asked  
where John is, and the world  
map is consulted to find the  
nearest ground.)

#### GRASP

John grasps the book.

John has the book. +  
The book is attached to John. +

John lets go of the book.

John doesn't have the book. +  
The book is not attached to  
John. +

#### INGEST.

John eats some cheese.

The cheese no longer exists. +  
John is less hungry. +

John drinks some water.

John is not thirsty. + If the  
water was not from a river,  
then also infer "The water  
no longer exists. " +

#### LOC

John is at the maple tree.

All the people near the maple  
tree get to know that John  
is at the maple tree. If  
John is holding anything,  
they get to know that. If  
they're holding anything,  
John gets to find out.

#### MTRANS

John tells Mary that Bill  
is in New York.

Mary knows that John told her  
that Bill is in New York.  
Mary thinks that John thinks  
that Bill is in New York.

#### PROPEL

John throws the baseball.

The baseball isn't near John.

John strikes Bill.

Bill is hurt.

John strikes the blueberry.

The blueberry is gone.

Mary pushes John into the  
river.

John is in the river.

#### PTRANS

John goes home.	John is home. +
John takes his books home.	John is home. John's books are at home. John knows his books are at home.
John leaves home.	John isn't home. +
John takes his books off the shelf.	John's books aren't on the shelf. +

#### 6.3 REACTIONS

**B**y far the single most prolific producer of inferences is MLOC, the state of knowing something. The number of things which can happen when a character acquires new knowledge is very large, so the MLOC-inferences are given a special name, reactions. In the following examples, John is the person who is reacting to new knowledge, so each event in the left-hand column is actually preceded by "John finds out that ..." The reactions are either relationships states, which are often preconditions to various actions, or new goals that will be immediately pursued.

#### ATRANS

Mary steals John's food. (anything John owns)	John dislikes Mary.
Someone steals John's car.	John is very unhappy.

#### CAUSALS

Bill will hit John if he doesn't give him the baseball.	John wants to give Bill the baseball.
---	---------------------------------------

Bill will hit John's  
friend Tom if John  
doesn't give Bill  
the baseball.

John wants to give Bill the  
baseball.

Bill will hit John's  
friend Tom if Mike  
doesn't give Bill  
the baseball.

John wants Mike to give Bill  
the baseball.

#### CONT

Bill has some food.

John is hungry (if he hasn't  
eaten for a while).

Bill has a baseball.

John wants the baseball.

#### HUNGER

John is hungry.

John wants not to be hungry.

#### LOC

John is underwater.

John wants to be out of the  
water.

John is near some food.

John is hungry (if he hasn't  
eaten for a while).

#### MLOC

Mary thinks John is  
brilliant.

John likes and trusts Mary.

Mary thinks Bill is in  
New York.

John thinks that Bill is in  
New York (if John trusts  
Mary).

Bill thinks John can't  
run very fast.

John wants to run fast (if  
he feels competitive  
towards Bill).

#### PROPEL

Bill hits John.

John doesn't like Bill.

#### REST

John is tired.

John wants to rest.

#### 6.4 ORGANIZATION OF MEMORY

**I**t is said that a liar has one of the most difficult memory tasks -- he must remember what lies he has told to each person he speaks to. A storyteller has the same problem. Since people in real life do not all believe the same things, anyone simulating them must keep track of each individual's knowledge. In addition, there must be some way of knowing what things are actually true, whether or not anyone believes they're true. This is why we maintain a separate memory for the storyteller. His knowledge is, by definition, accurate.

In the program's representation of memory, then, each set of facts is "indexed" under the name of the person who knows that set of facts, and there is one more set of facts indexed under the storyteller. Facts are, of course, represented as CD expressions.

Each such set of facts is further indexed by subject, corresponding to the idea, "What does John know about Bill?" A given fact may have several indices at this level. "Bill is in New York" has two indices, Bill and New York, so it would be stored with the things John knows about Bill and with the things John knows about New York. Each form of CD expression has a corresponding set of role-indices.

There is still one more level of indexing. This last index is either the ACT or STATE of the CD expression itself.

Explicit memory searches ("Does John think that Bill is in New York?") are simple and direct. We look for the set of things John knows, and within that, we look for the set of things John knows about either Bill or New York, and then we test whether the expression for "Bill is in New York" is listed there. In that example, the unknown element is the mode (or truth-value) of the entire expression. But when role-fillers are missing, as in the question, "Where is Bill?" the search depends on the role-fillers that are present. That is, the "location" role in the question has an undefined filler, so we can't use "location" as a role-index. This is why facts have more than one index, so that we can still retrieve them given only partial information.

Questions about generic items also present a problem. Suppose Joe Bear knows that there is some particular honey in the oak tree. In a memory search to answer the question, "Where is there any honey?" the retrieval mechanism is capable of matching the "particular" honey with the "generic" honey in the question.

TALE-SPIN's memory is actually stored in two structures. The first, called TRUEFACTS, represents those facts which are true at the present. The second, called OLDFACTS, represent things which were true at some time in the past. Questions specify whether they refer to the present or the past, and the appropriate structure is used. This is useful, for example, in determining whether to say "John went to New York" or "John returned to New York": if the representation for "John is in New York" is in OLDFACTS, then we use the word "returned." There

are also theoretical grounds for maintaining two structures. If I ask you where you are, it seems plausible that you can answer that quickly, and if I ask you if you've ever been in New York, that should take longer. Otherwise, if we used one structure for them all, and I asked you where you are, you'd have to go through all the places you'd ever been and see whether the time of your being there corresponded to the present time. "Let's see. I was born in Utah. Last week I was in Boston. Two months ago I was in Los Angeles. Today I'm in New Haven. That's it! I'm in New Haven."

Facts are moved from TRUEFACTS to OLDFACTS when they are contradicted. That is, if "John is in New Haven" is in TRUEFACTS and we find out that John is now in New York, we move "John is in New Haven" to OLDFACTS. For each kind of CD expression, there is a set of expressions which it contradicts.

The expression "X doesn't exist" contradicts all references to X, so when a character dies, or when an object such as food is consumed, all references to it are moved from TRUEFACTS to OLDFACTS.

#### EXAMPLES

To show what kinds of contradictions are detected by the memory system, we present some examples of memory input ("New") which contradicts facts previously in memory ("Old"), causing them to be moved from TRUEFACTS to OLDFACTS.

1. New: John picks up the rock.  
Old: John had dropped the rock.
2. New: John drops the rock.  
Old: John had been holding the rock.
3. New: John has the baseball.  
Old: Bill had the baseball.
4. New: John is in New York.  
Old: John was in New Jersey.
5. New: John isn't in New York.  
Old: John was in New York.  
Suppose Bill was with John in New York and he saw John leave.  
Bill doesn't know where John is going, but he knows at least  
that John isn't in New York any more. If he later wishes to  
talk to him, he'll know that he has to find him first.
6. New: John likes Mary.  
Old: John disliked Mary.
7. New: John thinks that Bill likes Mary.  
Old: John thought that Bill disliked Mary.
8. New: John is thirsty.  
Old: John wasn't thirsty.



Facts are simply deleted from TRUEFACTS when new, more specific information is learned. If "John isn't in New Haven" is in TRUEFACTS and we find out that John is in New York, we simply delete "John isn't in New Haven" since it is implied by the new item. For each kind of CD expression, there is a set of expressions which are implied by it in this sense.

On the other hand, less specific items are ignored. If we know that John is in New Haven and we then find out that he isn't in New York, we ignore this new information.

#### EXAMPLES

1. New: John picks up the rock.  
Old: Bill had dropped the rock.
2. New: John has the rock.  
Old: John didn't have the rock.
3. New: John is in New York.  
Old: John wasn't in New York.
4. New: John thinks Bill is in New York.  
Old: John thought Bill wasn't in New York.

The purpose behind the special handling of this input is to keep the memory consistent and non-redundant. The "present" states and acts are needed to pursue goals -- "Does Martha Bear know where any honey

is?" Remembering "past" events enables us to produce a better English translation -- "Mary returned the book to John."

## 6.5 CHEATING

*T*he purpose of indexing memory is to attempt to model how humans remember things. We could simply save all the facts in one gigantic list and search through it every time we needed a piece of information, but that's not only inefficient, it's dishonest: Some things are harder to remember than others. The context in which people access information must also be part of the memory model. The fifty-states problem [26] illustrates access context: It is much harder for people to list the fifty states of the US than it is to identify whether Missouri, for example, is a state. It should also be harder in the computer model.

It's not hard to imagine how TALE-SPIN's memory organization could be used to retrieve information like the list of the fifty states. That is, with additional indices, we could make it as easy to answer the question "Where are all the characters?" as it is to answer the question "Is Bill in New York?" This is always a problem with computer programs. You can program a machine to make illegal chess moves, but you don't, not if you're claiming that it really plays chess. That is, if your goal is to understand how people play chess, you won't achieve that goal by writing a program that cheats, even if it "wins" by doing so. The rules for chess moves are explicit. The rules for the organization of

human memory are anything but explicit, so it's much harder to know when you're cheating. The organization that the program now uses may still be too powerful, but further refinements require a theory of emphasis. For instance, "Bill is in New York" is more "memorable" than "Tom is in New York" if Bill is your six-year old son visiting his grandparents 2000 miles away in New York and Tom is a distant New York cousin. Both of those facts are indexed under the person, Bill or Tom, but only the first fact is indexed under New York, so that you could answer the questions "Where is Bill?" and "Where is Tom?" satisfactorily, but when asked "Who's in New York?" you would only remember that Bill is there.

Another reason we don't want the memory to be totally accurate is that we want to model people's response to misinformation. Accidental misinformation is related to a theory of forgetting -- John may forget that Mary was in New York and may believe that she's in Miami, or he may forget where she is entirely, or he may even forget who Mary is. TALE-SPIN does not include such a theory, but it does model deliberate misinformation, characters who lie to other characters. Since people both forget and lie, a program which purports to model people but cheats by using perfect knowledge all the time defeats its own purpose.

## CHAPTER 7

### A THEORY OF STORIES

#### 7.1 INTRODUCTION

**W**riters are an ingenious lot. To meet their employers' needs, they'll use a variety of techniques to produce stories. Sample orders: Write a story that has a part for Robert Redford. Write a story by next Tuesday. We need a 2500-word soft-core porn story for next month's issue. Write a "man vs. the elements" story. Write a disaster story. Rewrite this story to fit in 1976. Rewrite this story to fit in 2076. Write a story that we can film for under \$200,000. Write a story with no e's in it. Write a story about writing a story about writing stories.

In this chapter I'm going to discuss the various levels at which we understand and write stories and where those levels come from. Then we'll see the two principal storytelling methods used in TALE-SPIN, top-down and bottom-up, and how the Aesop-Fable generator works. Then

we'll look at what it means for a story to be coherent and interesting, how we might approach the problem of style, and how we can make stories more complicated.

## 7.2 STORY LEVELS

There are many levels at which we can look at a story, many areas of knowledge to which we can refer in describing it -- just think of all the books written about the works of Shakespeare. TALE-SPIN does a very modest job in this respect. Its levels are those of simple problem-solving, social relationships, geography, and the conscious construction of fables. It is not concerned with issues such as the function of myths and fables in the late 20th century, the visual effect of English written entirely in upper case, or the social impact of computers in the humanities, although one could discuss the program's output from each of those perspectives.

Where do these levels come from? How are they organized? Let's suppose that a writer starts a story with some idea in mind for the story. Morals are examples of ideas. What is the relationship between the idea and the "domain of interest" or main setting? Consider an imaginary story called "We climbed Mt. Everest." The domain of interest includes DELTA-PROX; the "problem" of the story is getting to the top of the mountain. Beyond that level, what kind of story is it? We could describe it, in increasing order of abstraction, as a story about

climbing Mt. Everest, a mountain-climbing story, an outdoors adventure story, a story about man conquering nature, a story about the conflict between man and nature (winner unspecified), and a story about conflict. Which of these do we want to call the idea of the story? We seek the most useful distinction; all of them are valid to some degree, but the notion we want to capture is this: A writer sits down to write the story, "We climbed Mt. Everest," and he says to himself, "Today I'm going to write a story about \_\_\_\_" It seems improbable that he would fill in the blank with "conflict," because that's the ultimate idea behind all stories. It seems equally improbable that he'll use "climbing Mt. Everest," because that's so specific. So it's somewhere in between.

To clarify the notion of "most useful distinction," let's consider an analogy. How can we describe, in increasing order of abstraction, the McDonald's restaurant at the corner of Main and High? Well, it's a hamburger joint, a restaurant, a pay-for-service facility, a building that people use, a building, and a physical object. Similarly, we can describe the MidTown Cinema as a art-film movie theater, a movie theater, a pay-for-service facility, a building that people use, a building, and a physical object. The question is, what is the most useful distinction between the McDonald's and the MidTown Cinema? The answer, of course, depends on who's making the distinction. For a customer, it's very important that one is a restaurant and the other is a theater. For a carpet salesman, however, that doesn't matter; the important thing is that they are both buildings that people use. A

landscape artist may not care that people use the buildings; all he cares is that they are buildings. And maybe Martians and computational linguists care that they are physical objects.

The point of this example is that the most useful distinctions aren't always going to be on the same level. While we'd like to be able to say that writers start with the abstraction exactly three levels above the literal level of the story, it just isn't going to be that easy. A writer's collection of ideas aren't all at the same level. For all its exciting scenes, Moby Dick is not an outdoors adventure story. That is, if you tried to write it starting at that level, you'd be missing a lot. For example, one analysis of Moby Dick says that the great white whale, which is the symbol of Evil Incarnate, is white because Melville saw white as the color of an empty universe, in which it can clearly be seen that nothing exists, that there is no God, and therefore all is hopeless and evil. In darkness, by contrast, there is always the hope that there's something out there we can't see, something that might be good, so black is not the color of evil. Now if you believe even a part of that analysis, you'll realize why "outdoors adventure" is too low a level from which to start. The fact that the whale is white is a crucial one, not something that could be made at random, and it comes from a level like "good versus evil," which is much higher than "outdoors adventure." So two stories which may look alike may in fact be generated from entirely different levels of ideas.

The theory, then, is that the level of the idea of a story is the



highest one in which decisions are made that influence the story. That is, we hypothesize a hierarchy of ideas, each level more abstract than the next. Each level contains information which can be used as a basis for making decisions in the domain of interest. To return to the Everest example, suppose we wanted to make it not simply "outdoors adventure" but "man versus nature." Then we might make the mountain seem as if it bears malice toward the climbers by including particularly violent snowstorms, or by making the psychological nature of the climbers such that the constant difficulties began to wear down their morale, or even by having one of the characters anthropomorphize the mountain itself, cursing it by name. The climbers who succeeded could be made to feel a sense of personal triumph. If we were telling only the "outdoors adventure" story, there'd be no reason to put any of those things in.

If stories are about conflict, then the ideas are of the form "force vs. force." What forces are there? At the very abstract level, there's the force of good and the force of evil. Some writers view the force of nature as neither good nor evil, but rather indifferent: wind, sea, mountains, disease. Time is a force. Races are conflicts with time, whether on foot or in expensive cars. The passage of time is a force: physical weakness, opinions that become old-fashioned. Man is driven by many forces, beliefs which compel him to act one way or another. Conflicts can arise between your own beliefs, between your belief and someone else's, or between your belief and everyone else's. The urge to learn is a force, as is the pursuit of physical comfort and pleasure, which may or may not come under the headings of good or evil.

Why choose one level and not another? No reason. You can start anywhere. The choice is entirely outside the writing of the story. It's like asking why Melville chose to write a story about good and evil. There probably was some reason, but it didn't matter as far as the story was concerned. The problem in writing stories by computer is not how to choose the level, but rather how to proceed once the level is chosen.

As a final note, I should say that an AI approach to literary theory is very different from that of, say, Northrop Frye in his Anatomy of Criticism [10], which deals with matters such as myth and mimesis. These concepts are far beyond the current bounds of AI, but if our view of the nature of stories sounds naive, as if we were oblivious to the literary theorists, it's because we're dealing at a much more practical or pragmatic level. Polti's Thirty-Six Dramatic Situations [19] may be of greater use to us in the near future than the work of Frye.

### 7.3 THE TWO METHODS OF STORYTELLING IN TALE-SPIN

**T**ALE-SPIN includes a simulator of the real world: Turn it on and watch all the people. The purpose of the simulator is to model rational behavior; the people in it are supposed to act like real people. To do that, we have to model various kinds of human thinking, how people relate to each other socially, for example. What happens when the simulator runs may or may not be interesting, and since interest is a

criterion for stories, we may or may not get a story by simply watching the simulator run.

How do you make it interesting? You fix it in advance. You rig the world so that if people do behave rationally, they'll do some interesting things. So sitting on top of the simulator, if you like, is a program which knows both about stories and about the world model. It models a writer who has something in mind that he wants to tell a story about. He starts with some message or moral, chooses the problem domain of interest, and rigs the world accordingly. This is what I call the top-down approach. This is TALE-SPIN's mode 3.

The other method, the bottom-up method, is to let the reader choose the domain of interest. A selected character is given a problem, and the simulator starts by having him solve that problem. The solution may cause other problems to arise, and they will also be solved. This is intended to model someone who makes up a story on the fly, rather than someone who plans everything out in advance. This is TALE-SPIN's Mode 1. (Mode 2 is Mode 1 with prettier output.)

Let's see how TALE-SPIN uses a particular moral to rig the world.

#### 7.4 THE AESOP-FABLE GENERATOR

**N**ever trust flatterers. So says Aesop in the fable called "The Fox and the Crow." There are lots of ways to get a program to tell that

story, the easiest being simply to have the computer read it in and print it out. Another method, used by Klein (see section 1.3), is to write some code which will produce each part of that particular story when the whole program runs. Such programs, however, are limited to telling exactly those stories that have been put into them, whole or piecemeal. But there are many stories that have "Never trust flatterers" as their moral. How can we get a computer to write them?

The first part of the answer is to understand what that moral means. I interpret it as follows. "Never do X" means that if you do X, then something "bad" will happen. "A flatters B" means that A says something "nice" to B, but is insincere, doing it for some ulterior motive. Since a consequence of saying something nice to B is that B will become more kindly disposed toward A, then it's reasonable to assume that B's kind disposition toward A will enable something to happen which is "good" for A. Putting this all together, we predict that A has some goal which requires that B be kindly disposed toward A, so A says something nice to B, B reacts accordingly, something happens which causes A to achieve his goal and also causes B to suffer.

Looking at the domains that are in TALE-SPIN, we can see that DELTA-CONTROL can be the "interesting problem": A's goal is to get some object which B has. The transfer of ownership is simultaneously "good" for A and "bad" for B. That much of the story -- a conflict over possession of some object -- is built into the program in the following way. The object must be something that both A and B would want. In

that category are "foods" and "properties," as far as TALE-SPIN is concerned. With each kind of character (bear, bird, boy, ...) is associated a list of foods and a list of objects ("properties") which the character might own. The program which tells the story asks the reader for two characters (A and B) and attempts to tell the "Never trust flatterers" story by finding some food or property in common. If it fails to find one, it can't tell that story. If there are any, it selects one at random and then creates the characters and the object, giving the object to B. It asserts that A is dishonest and that B is vain. Then it gives A the goal of being near B. The simulator then takes over. The first planning structure to be used is DELTA-PROX, since it is the appropriate method for achieving A's first goal. When A gets near B, A notices the object and since it is either one of his foods or one of his possessions, he desires it, which invokes DELTA-CONTROL, and so on.

The only way this method will fail to tell a story is if there are no objects in common between the two types of characters that the reader specified. One way to guarantee a story is for the reader to specify two characters of the same type, two bears, for example.

What did we learn from doing this? First, like all extensions to the program, it pointed out unanticipated needs in the world model. One of them (see Chapter 8, story #5) was that in setting up the world, you had to be careful not to activate any automatic goals. In that story, when we gave the crow the cheese, the simulator responded by making the

crow realize that it had the cheese, and since that was a food that crows eat (in the story), the issue of hunger was automatically raised. Memory was checked to see whether the crow had recently eaten; it had not, so a goal of eating the cheese was asserted, and the crow, having all the requirements specified by SIGMA-HUNGER, ate the cheese. When the fox came over later on, there wasn't any cheese for him to notice and become hungry for, and the "story" ended right there -- no new goals had been asserted.

The second thing we learned was that altering the characters did not drastically alter the resulting story. They all came out looking like simple-minded variations on "The Fox and the Crow," which, of course, they were. It was clear that using the DELTA-CONTROL problem domain was the significant factor: Once you've decided that someone is going to use flattery to trick someone else out of some possession, you're very limited in the number of different stories you can tell. Variety would stem from more and more elaborate tricks.

The third thing we learned was that if you made the two characters competitive with each other (instead of one dishonest and the other vain), you got an entirely different story. This is how "Joe Bear and Jack Bear" was produced (see Chapter 2). The trick was for one character to challenge the other into doing something which would require him to leave, so that the first character could grab the object and take off. It was a very small change in the input, in the way the world was set up, but it produced a very large change in the output, in

the resulting story. The point of the story wasn't even about flattery. It had its own moral: Never trust challengers. This suggests that one way to organize the morals of stories, a problem I discuss in the next section, is to consider what the "rigging" requirements are, the conjecture being that you should consider as a single group those morals of stories which have similar rigging requirements.

The final thing to be gained from studying the Aesop model is to consider whether the process of selecting a problem domain can be automated, deduced, as it were, from the moral. A writer, for example, could tell a "never trust flatterers" story about DELTA-KNOW (A tricks B into telling him some state secret), PI-MONEY (young greedy John sweet-talks rich old widow Mary into marrying him, grabs the money and runs), social power (ambitious junior exec befriends corporate president, only to use his new status to defeat the president in a power struggle), and so on. It may be that human experience is so rich that any problem domain will suffice, given a moral to start with. If that is the case, then the choice of domain can't be determined by looking at the moral -- it's too general. We can choose a domain at random and then figure out the plot simply by knowing a great deal about the domain. The choice of domain may even alter the emphasis of the story to the point of obscuring the moral. There are lots of stories in which the moral is, shall we say, less than prominent, stories where the emphasis is at some lower level. The Towering Inferno, a "disaster" film about a skyscraper that burns up because of shoddy construction, will not be remembered for its moral -- evil begets more evil, or



whatever -- nor for its similarity to the story of the Tower of Babel, but rather for its special effects.

### 7.5 COHERENCE

**C**oherence is the measure of how much sense a text makes, how well it corresponds with our model of the real world. Consistency dominates coherence -- there is a level of understanding at which talking bears are anomalous, but we're accustomed to stories which extend the world model with anthropomorphic devices such as this, so it's consistent with a large body of literature. In general, a coherent text is one that obeys the laws of some causality, some causal context, and they come in many shapes. It would violate a story-level causality to give the power of speech to bears and not, say, to wolves in the same story. A much more basic causality would be violated if a character lost the power of speech between Chapter 1 and Chapter 2.

We've seen a lot of planning-causalities in this work, and it would violate the knowledge in SIGMA-HUNGER if Joe Bear satisfied his hunger by counting to thirty. At one level, that's coherent; we know what all the words mean. At the other level, however, it makes no sense; we don't see the causal connection. At yet another level, it might be sensible, as part of a magical formula, a dream, life on a distant planet.

TALE-SPIN's prime concern is coherence at many levels of

understanding, with particular emphasis given to solving physical problems and interacting socially. By insisting on this kind of coherence, TALE-SPIN could never produce a story like Kafka's "Metamorphosis," in which a man wakes up one day as a giant cockroach. This is certainly anomalous, however good a story it may make. No explanation is ever given, nor does it happen to anyone else. If we call it an "acceptable" story, we then need a theory that tells us what kind of anomalies are permissible. Would the story have been as good had he changed into a parakeet? Two cockroaches? A grandfather clock? The particular choice certainly requires knowledge about cockroaches. That is, Kafka attributes certain characteristics of cockroaches to the man in the story. TALE-SPIN lacks such knowledge, and therefore has no basis to judge. Since random anomalies can cause arbitrary damage to the coherence of a story, we avoid them.

#### 7.6 INTEREST

**W**e've seen that some anomalies are interesting, and some aren't. What do we mean by "interesting"?

Readers are interested in characters and situations they can relate to, whether by identifying with them or by recognizing people they know. Biographies are often interesting. One's own biography, I would imagine, would be even more so.

But you can relate to a story on many levels. You may relate to

biographies on a literal level, but stories in which the kinds of things which happen to the characters are the kinds of things which happen to you are also interesting. On one level, the point of Aesop's fable about the fox and the crow is that the crow should not have trusted the fox. Being neither a fox nor a crow, I can't relate to that at the literal level. But another level, the point is that we shouldn't trust flatterers, which is much easier to relate to.

However, that's not enough. Most stories about brushing your teeth are not interesting, no matter how strongly we relate to them, no matter how many times you've brushed your teeth. "Joe Bear was hungry. There was a jar of honey right next to him. He ate it. The end." That's not very interesting, either. These non-stories lack an element of difficulty, a problem whose solution is not immediate and obvious, a situation which is not entirely predictable. If the crow in the fable had been smart enough to figure out that the fox was tricking him and had refused the fox's offer (without opening his mouth!), the story might not have been as interesting. In fact, there is such an Aesop fable, "The Fox and the Hen." Perhaps it is surprising -- and therefore interesting -- that the hen is smart enough, or perhaps it's the joke at the end of fable. The hen, supposedly ill, refuses to come out and say hello to the wily fox, claiming that to do so "would be the death of me."

TALE-SPIN's mode 3 creates the world in such a way that the main character's problem is not easily solved, on the grounds that things

that go wrong are interesting. If you include a boring, low-level detail in your story, your reader will assume that something is about to go wrong, that you mentioned the detail only to set up the situation more clearly. "John chewed the steak. Suddenly he turned purple." Map details (see Chapter 9) are also in this category. If you're telling a story about a trip and you mention a specific detail about the route, your reader expects some consequence. "I was driving to work this morning. After I got off the highway, I had just made a left turn onto Main Street when ..." Your reader expects

"... I got a flat tire"

or "... I hit a police car"

or "... I realized it was one way the other way"

or "... I saw Joe Schmidlap"

or "... my carburetor started working again."

The last two examples point out that it isn't simply things that go wrong that are interesting, it's also things which are unusual. (For some lucky people, that's a superset.) We expect that Joe Schmidlap is somebody famous or was presumed dead or is never seen in daylight or is in some other way unusual. We are surprised that the carburetor fixed itself.

If you had continued with "... I parked the car and walked to my office," your readers will wonder why you said all this. It was coherent but not interesting.

Interest and coherence are fairly independent, except for the case

of the totally incoherent text, which is almost certainly uninteresting as well.

In general, then, a story is about a problem and how it gets solved. The domain of the problem is called the domain of interest. Problems in other domains are not included in the story. They exist, but their solutions aren't necessarily interesting or important. If the domain of interest is acquisition of social power, for instance, as in "How John became mayor," then it's unlikely that the story will include problems of getting food. It's not that would-be mayors don't eat. It's just that the domain of SIGMA-HUNGER is far removed from the domain of interest. Problems in the domain of interest either are always hard or are deliberately made difficult.

The solution can't be too easy, which is why the following is not a story: "Joe Bear was hungry. There was a jar of honey sitting right next to him. He ate it. The end."

The film Gambit is another good illustration of this point. A man is trying to persuade a woman to join him in a jewel robbery. The screen goes fuzzy, then we see the woman taking part in the heist, so we assume that the man was successful in persuading her. The heist is very elaborate, but it goes off without a hitch, very slick and professional. And dull. But then the screen goes fuzzy again, and we see the man still talking to the woman about the planned robbery. He has been explaining all along how he hopes the robbery will go. So the woman

agrees, and in the actual robbery, everything goes wrong, of course, and the resulting story is funny and very interesting. It is a very non-standard jewel-heist movie.

If stories are about problems, what kinds of stories can TALE-SPIN write? Of all the problem domains, TALE-SPIN knows the most about the delta-acts, DELTA-KNOW, DELTA-PROX, and DELTA-CONTROL. Those are general problem-solving techniques, and in any story in which someone has to get from one place to another, for example, DELTA-PROX is applicable. Since it is a component of planful behavior, it's part of any story about planful behavior, which is to say, any story about humans. But DELTA-PROX is not always in the domain of interest; not all stories are "about" journeys. When journeys occur, they may be referred to briefly if at all -- the reader can infer the details if necessary. So TALE-SPIN is limited to stories whose domain of interest is included in the model. It cannot yet tell the story "John wanted to be mayor," therefore, because it does not yet model the domain of social power. If we had the planning structure for achieving social power, we could tell that story, because we'd know what the requirements are for obtaining social power, and therefore what things could go wrong.

But we must restrict the things that go wrong to the domain of interest. Consider this story:

John wanted to be mayor. He knew he needed votes, so he started talking to the people on his block. The lady next door said she'd vote for him if he could lower food prices, so John decided to have a talk with Mr. Jones, the grocer. But John didn't know where Mr. Jones's store was, and he didn't have a car, so he ....

That last sentence doesn't fit -- the problem there is at too nitty-gritty a level. The problem in a social-power story should be on the social level. We expected to hear that John went around convincing people that he was competent and trustworthy. Instead, we got DELTA-KNOW and DELTA-PROX problems, which seem inappropriate.

The inappropriate continuation can go in the other direction, too, towards higher-level problems.

Karen was sitting outside, reading a junk novel. She wanted to finish the book, but it soon became too dark to read. While thinking about finding some light, she wondered why it gets dark at night. She recalled that it was not due to the sun's being on the other side of the earth, but rather to the fact that the stars were receding. Karen wanted to find out the metaphysical implications of an expanding universe.

We've come a long way from the goal of finishing the junk novel. Too far, in fact.

In summary, an interesting story deals with a central problem that the readers can relate to, a problem which is not easy to solve. Unusual things are interesting, as are things that go wrong, but they must be closely related to the central problem, belonging either to the same domain or to the domain of some closely related subproblem.

#### 7.7 STYLE

"*T*he way in which something is said or done, as distinguished from its substance." That's a dictionary definition of style. I understand "substance" to mean the idea of the story and the problem domain. Even



after those are settled, there are lots of choices still available.

Which does the writer choose? It's a matter of style.

The raven said, "Never again."

"Eighty-seven years ago our forefathers ..."

So the Big Bad Wolf cried, "Little pig, little pig, let me in, let me in!" "I think not," replied the pig.

Rhett turned to Scarlet and said, "Frankly, my dear, I don't give a hoot."

Why do authors choose the words they do? Sometimes we know: If the story follows a form (such as rhymed couplets), if there's some particular emotion involved, or if the speaker has a particular kind of personality (pompous, crude, earnest), then we may know enough to prefer one phrase over another one which means the same. But that won't explain why Dickens used these exact words: "It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness..." It is not enough to say that it is anaphora; what we need is a theory that explains why he chose to use that stylistic device here, why he used the words he did within these four phrases, why there are ten such phrases and not, say, eight or twelve.

Choosing particular words and phrases is like choosing anything else. You've got to know what the alternatives are, what the preconditions are, what the side-effects are, and how well a particular choice fits in with your goals. For example, before I use the word "susurratation," I should wonder whether my readers will know what it means. If I did use it, then a side-effect would be that I probably

shouldn't use it again in the same work -- it stands out too much. A precondition on the use of the word "meaning" is that the reader know which sense of the word I intend. To mix metaphors is to pursue conflicting linguistic goals.

As a problem domain, linguistic style, if I may call it that, is certainly as rich as the domain of everyday problems, DELTA-PROX and so on. When we describe a word as obsolete or regional or archaic, we're describing intrinsic properties of the word, properties within the domain of linguistic style. But when we describe a word as apt or inflammatory or humorous, we're saying how this domain relates to other domains -- aptness is not an intrinsic property of a word.

Linguistic style is only one kind of style. There's also a style of plots. Stories in which nearly everything is a problem are what we might call "Trials and Tribulations" stories, T&T for short. Most of the "Joe Bear" stories in this thesis are T&T stories by accident rather than by design, because the worlds in which those stories begin are very sparse. Almost none of the problems that arise are already solved. In this sense, T&T is the default style. "Wheel spokes" is another style of story, where the characters, having carried on in their individual subplots for most of the story, all meet in one place at the very end of the story. Sometimes the final scene is the climax of the story. At other times, the action is all over by the time they all meet, as in the film Dinner at Eight.

Murder mysteries have their own stylistic rules. The identity of

the murderer in a murder mystery is not revealed when the murder is described. If it were, it would be understood, it might be interesting, but it does not conform to the style, because at one level, a mystery story is a game between the author and the reader. The author withholds a key piece of information, and the reader tries to guess it.

Stream-of-consciousness stories have their own style; classical narratives have theirs, in medias res. Most of the Aesop fables, in fact, are stylistically similar -- personal confrontations between animals, often utilizing an everyday problem (hunger, thirst, greed, ...) as the means for illustrating some lesson about humanity (ironically enough).

The style of soap operas is more than simple T&T -- soap operas never end. They perpetuate themselves by introducing, with abandon, problems whose resolutions create more problems. The world never reaches a state of equilibrium. Many TV shows, on the other hand, do exactly the opposite: Equilibrium is reached every thirty (or sixty) minutes, which has the advantage that re-runs can be shown in any order. Episodes are as interchangeable as light bulbs. Either they concern problems which aren't serious to begin with, or they use a deus ex machina to fix things up in a hurry. That, in particular, wouldn't be too hard to do with TALE-SPIN -- since the goal structures specify where the story is, and since we can create solutions to any problem ("Suddenly Joe spied a beehive in the distance ..."), we could solve all the problems. The real task would be not to outrage the reader, as is

always the danger with deus ex machina.

Sometimes style is predicted by a sense of balance, "poetic justice," which is certainly a valuable heuristic both in understanding a story -- by predicting future events -- and in writing a story -- by limiting our future choices. For example, the style of disposing of the villain at the end of a story can be predicted by his actions during the course of the story. Readers can guess whether he'll die of old age, or be shot by the police, or be taken care of by a great white shark, or be standing alone atop a giant, spherical, gas storage tank, screaming, "I'm on top of the world, Ma!" as the tank blows sky-high. That's style. We discuss it more in Chapter 13.

#### 7.8 COMPLEXITY

A story is a slice of time in the history of a world. Starting at an arbitrary point presents the difficulty that the reader doesn't know what's been going on. Without help, he may not be able to understand the story. The help is called "setting." "Once upon a time there was a brave lad named Jack. Etc., etc., etc. One day, he ..." Fairy tales use a very simple mechanism to present the setting. They surround it with the phrases "once upon a time" and "one day." These cues let us know what's part of the story proper, and what's part of the background information. Likewise, the phrase "they lived happily ever after" lets us know that the story is over, that the main conflict has been

resolved, but that the world goes on.

TALE-SPIN is close to the light-bulb paradigm described in the last section. The main character is given some problem to overcome. During its resolution, he may encounter other problems or cause problems for other characters. The basic control structure of the simulator calls for each new problem to be solved, but eventually this process terminates. You can then use the same characters and tell another story by giving one of them a new problem. Everything that was true at the end of the first story is true at the beginning of the second, and so on.

But how would we produce a long story that wasn't merely the agglutination of episodes? The easiest way is to give the main character a very hard problem, one that will take a long time to solve. TALE-SPIN can produce such stories, although the longest stories were disasters of single-mindedness (see Chapter 8).

Ultimately, the length of a story depends on the complexity of the world model. Since TALE-SPIN's world model is not very complex, its stories are not very long. In Chapter 3, I mentioned possible additions to the planning structures, and these would increase the model's complexity. Goals of larger scale -- becoming president, for example -- would add to the model. Each addition requires the acquisition, specification, and encoding of new areas of knowledge, some of which are more germane to stories than others. Social and psychological theories would certainly be relevant. A sophisticated understanding of the

process of visual recognition, on the other hand, is a legitimate goal for AI, but it may only marginally improve a program's ability to tell stories.

## CHAPTER 8

### MIS-SPUN TALES

While one can present a theory as if it had sprung full-blown from one's head, it is often true that the theory was developed piece by piece. Such is the case with some of the theories behind TALE-SPIN. It was difficult to anticipate all the needs of a story-telling program. How many ways are there of getting from one place to another, for instance? I used some small set that seemed to cover most cases, but when they proved inadequate, I added more.

Sometimes the theory gets too far ahead of the implementation or concerns problems which will never arise in the cases I want to handle. Building a new AI system is a race between theory and implementation.

One of the delights of working on a story generator has been that bugs, inconsistencies, and inadequate theories frequently exhibit themselves as ridiculous, often amusing stories. Such things are called "horror stories," and they can be very instructive: they justify much



of the detail of the program. I only found out that I had to add such-and-such a test when the program produced an absurd story because that test was missing.

These stories were all produced by early versions of TALE-SPIN. Since they're not horror stories about the English generator (see Chapter 12), I'll use hand translations.

\*\*\* 1 \*\*\*

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe threatened to hit Irving if he didn't tell him where some honey was.

Joe has not understood that Irving really has answered his question, albeit indirectly. Lesson: answers to questions can take more than one form. You've got to know about beehives in order to understand that the answer is acceptable. Also, it's polite to give some details when you answer a question. ("Do you know what time it is?" "Yes.") So now Irving says that there's honey in the hive and that so-and-so (a bee) owns the honey.

\*\*\* 2 \*\*\*

One day Joe Bear was hungry. He asked his friend Irving Bird where some honey was. Irving told him there was a beehive in the oak tree. Joe walked to the oak tree. He ate the beehive.

A further refinement is to unscramble an acceptable answer in the proper fashion, remembering a little better what the original question was.

\*\*\* 3 \*\*\*

In the early days of TALE-SPIN, all the action focused on a single character. Other characters could respond to him only in very limited ways, when they were asked direct questions, for example. There was no concept of "noticing": If people walk into your room, they needn't always announce their presence. You see them. The following story was an attempt to produce "The Ant and the Dove," one of the Aesop fables.

Henry Ant was thirsty. He walked over to the river bank where his good friend Bill Bird was sitting. Henry slipped and fell in the river. He was unable to call for help. He drowned. That wasn't supposed to happen. Falling into the river was deliberately introduced to cause the central "problem" of the story. Had Henry been able to call to Bill for help, Bill would have saved him, but I had just modified DO-MTRANS so that being in water prevents speech, which seemed reasonable. Since Bill was not asked a direct question, he didn't notice his friend drowning in the river. "Noticing" is now an inference from change of location, described in section 11.18, and Bill sees Henry in the river, deduces that Henry's in danger, and rescues him.

\*\*\* 4 \*\*\*

Here are some rules that were in TALE-SPIN when the next horror occurred. If A moves B to location C, we can infer not only that B is in location C, but that A is also. If you're in a river, you want to get out, because you'll drown if you don't. If you have legs, you might be able to swim out. With wings, you might be able to fly away. With friends, you can ask for help. These sound reasonable. However, when I

represented "X fell" as "gravity moved X," I got this story:

Henry Ant was thirsty. He walked over to the river bank where his good friend Bill Bird was sitting. Henry slipped and fell in the river. Gravity [having neither legs, wings, nor friends] drowned.

Now "X fell" is represented with PROPEL, not PTRANS, that is, as "the force gravity applied to X," and the inferences from PROPEL are not the same as for PTRANS.

\*\*\* 5 \*\*\*

The inclusion of awareness meant that I couldn't set up the stories the way I used to.

Once upon a time there was a dishonest fox and a vain crow. One day the crow was sitting in his tree, holding a piece of cheese in his mouth. He noticed that he was holding the piece of cheese. He became hungry, and swallowed the cheese. The fox walked over to the crow. The end.

That was supposed to have been "The Fox and the Crow," of course. The fox was going to trick the crow out of the cheese, but when he got there, there was no cheese. The fix was to assert at the beginning that the crow had eaten recently, so that even when he noticed the cheese, he *didn't become hungry*.

\*\*\* 6 \*\*\*

Before there was much concern in the program about goals, I got this story:

Joe Bear was hungry. He asked Irving Bird where some honey

was. Irving refused to tell him, so Joe offered to bring him a worm if he'd tell him where some honey was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was. Irving agreed. But Joe didn't know where any worms were, so he asked Irving, who refused to say. So Joe offered to bring him a worm if he'd tell him where a worm was....

Lesson: don't put a goal on the stack if it's already there. Try something else. If there isn't anything else, you can't achieve that goal.

\*\*\* 7 \*\*\*

Here are some more rules. If you're hungry and you see some food, you'll want to eat it. If you're trying to get some food and you fail, you get sick. If you want some object, try bargaining with the object's owner. Innocuous, right?

One day Henry Crow sat in his tree, holding a piece of cheese in his mouth, when up came Bill Fox. Bill saw the cheese and was hungry. [Hunger is now on his goal stack.] He said, "Henry, I like your singing very much. Won't you please sing for me?" Henry, flattered by this compliment, began to sing. The cheese fell to the ground. Bill fox saw the cheese on the ground and was very hungry. [Hunger is about to be added to his goal stack.] He became ill. [Because hunger was already on his goal stack, he couldn't get the food, so he gets

sick.] Henry Crow saw the cheese on the ground, and he became hungry, but he knew that he owned the cheese. He felt pretty honest with himself, so he decided not to trick himself into giving up the cheese. He wasn't trying to deceive himself, either, nor did he feel competitive with himself. But he did dominate himself, and was very familiar with himself, so he asked himself for the cheese. He trusted himself, but he remembered that he was also in a position of dominance over himself, so he refused to give himself the cheese. He couldn't think of a good reason why he should give himself the cheese [if he did that, he'd lose the cheese], so he offered to bring himself a worm if he'd give himself the cheese. That sounded okay, but he didn't know where any worms were. So he said to himself, "Henry, do you know where any worms are?"

But of course, he didn't, so he .... [And so on]

The program eventually ran aground for other reasons. I was surprised it got as far as it did. The fix was to make the inference that dropping the cheese results in loss of ownership.

\*\*\* 8 \*\*\*

The last horror story is also related to goals. When you achieve a goal, it's no longer a goal. If you tried but failed to achieve a goal, it's also no longer a goal. Before I had the program make a distinction between those two cases, it produced the following story. Due to its length, I'll write parts of it in conversational mode (see Chapter 12). If you get lost while reading it, that's okay: you're supposed to.

Once upon a time, there was a bear named Joe who lived in a cave in the mountain. One day he was hungry, so he went to see his friend Irving Bird. "Irving," he said, "Will you tell me where I can find some honey?"

Now Irving liked Joe a little, but he wanted to play a trick on him. "Certainly. Arthur Bee has some over in his beehive in the oak tree." He was lying. There wasn't any such bee, although there was an oak tree.

Joe, being an honest bear, decided to ask Arthur for the honey. "I guess he'll like me enough to give me some honey," he thought. So he walked across the valley to the oak tree. Lo and behold, no beehive. Joe was pretty mad at Irving for lying to him about Arthur. He decided Irving might tell him where Arthur really was if he brought him a worm. So he went back to Irving. "Say Irving, would you tell me where Arthur really is if I brought you a worm?"

"Oh sure," replied the bird.

"Good," thought the bear. He went off to get a worm he knew about. He brought it back and gave it to Irving.

Irving was very hungry and ate the worm. "Dumb bear," he said to Joe.

Joe was mad. He couldn't find out where Arthur was to ask him about the honey. "Ah! Maybe he can bring the honey to me," he thought. He was still mad at Irving, and knew he couldn't trust him to ask him to do a favor for free. He couldn't think of any good reason why Irving should help him, so he decided to offer him another worm. "Look, Irving, if I bring you another worm, will you ask Arthur if he'll bring me the honey?"

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so he gave up on that idea. "If I can't do that, then Arthur won't bring me the honey. Maybe he will if I bring him a flower. Bees like flowers, right?" [Oh no.] "First, I've got to find out where Arthur is."

"Say Irving, would you tell me where Arthur is if I brought you a worm?"

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so he gave up on that idea. "If I can't do that, then Arthur won't bring me the honey. Maybe I can offer Irving a worm to go ask Arthur."



"Say Irving, would you ask Arthur if he'll trade me his honey for a flower? I'll bring you a worm if you do."

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so he gave up on that idea. "Damn. Maybe I can steal the honey from him. I'm really hungry now. Hmm. Arthur's in the beehive. I'll pull him out. Where is this bee anyhow?"

"Irving, would you tell me where Arthur is if I brought you a worm?"

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so heI can trick him into leaving the hive."

"Irving, would you tell me where Arthur is if I brought you a worm?"

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so he gave up on that idea. "Maybe I can get Irving to ask Arthur to leave the hive."

"Irving, would you ask Arthur to leave his beehive? I'll bring you a worm if you do."

"Of course," said the bird. But Joe didn't know where any more worms were, and he wasn't going to try to persuade Irving to tell him, so he gave up on that idea. "Maybe if I got Irving to bring Arthur a flower...."

## CHAPTER 9

### MAPS AND BLUEPRINTS

#### 9.1 MAPS

"Joe Bear was sitting in his cave one day when he decided to get some food. He knew there were some blueberries growing near the oak tree. He reached over and grabbed them." "Martha Worm wanted to visit her cousin Bertha Worm who lived on the other side of the river. She crawled across the river and up the bank." "John was crossing the street. A car, out of control, headed straight for him, the desperate driver honking the horn to warn John. John continued walking, straight into the car."

Any reasonable model of the world has got to deal with the problem of representing space, the physical world. The stories above are all anomalous because they contradict our knowledge of the physical world: being in a cave is not the same as being outside; worms cannot crawl on water as they can on land; you notice your immediate surroundings,

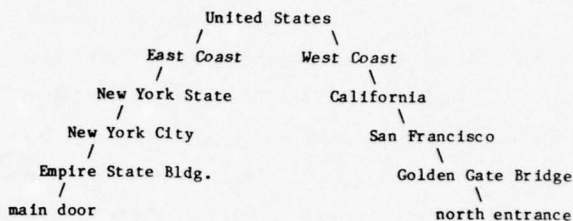
sights, sounds, smells, and so on. We need to represent locations, distances, routes. Since we are trying to model humans, we can rule out the simple expedient of giving every fixed location a set of Cartesian coordinates and using some geometry to compute distance along straight-line paths.

Our first task is to represent locations. It helps to have an example in mind, so consider the following problem: I'm sitting at my office desk in Room 331 on the third floor of Dunham Lab at Yale and realize that I left some important papers at home on my desk in Room 2813 on the fifth floor of the Hall of Graduate Studies (HGS), which is about six blocks away from Dunham. I formulate a plan to get them.

I define a map as the conceptual representation of a region of fixed size, containing elements of the same scale. A map of Connecticut, for example, has a region (submap) called "New Haven," but my office desk would certainly not appear on that map: the desk is too small relative to the city of New Haven and other regions on the state map. The map of New Haven might contain a submap called "Yale Campus," but my office desk is still too small. Within the campus map is a map of Dunham Lab, and within that is a map of my office, on which, finally, the desk appears. I call this nesting of maps the map hierarchy.

Of the many possible ways to represent how people might picture the physical world, I use this one because it seems to explain why people who are asked the distance from the main door of the Empire State Building to the northern entrance to the Golden Gate Bridge seem to

relate it to the problem of finding the distance between the east coast of the US and the west coast. The map theory says that when people picture two points at the same time, to estimate distance, for instance, they search for a map containing both points, a map on which the points appear as submaps, or as submaps of submaps. Probably no one has a map which includes both the main door of the Empire State Building and the northern entrance to the Golden Gate Bridge as immediate submaps, so they back up, as it were, one level at a time, perhaps in parallel, until they find a large enough map.



How do people know the distance from one coast to the other? This fact may actually be stored; most people think they know the width of the country. (If not, they try to estimate it in a number of ways, especially by considering how long it takes to get from coast to coast.)

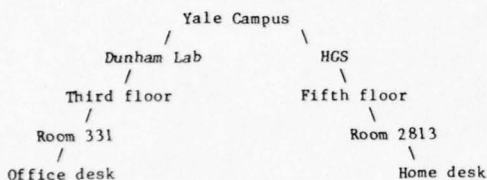
There is no rule specifying how much information a map contains. The detail of a map, that is, how many regions in it are themselves maps, can depend on knowledge gained from experience. My map of the state of Connecticut has some very detailed submaps, but if you ask me

to picture Utah, I imagine a vaguely rectangular area, perfectly blank, without a single city on it. (Of course, I might recognize the name of the state capital of Utah if I came across it, but that's another issue.)

How big do maps get? Many people probably know about the continents, the planet, the solar system. Others, more highly trained, know about the relation of our solar system to the Milky Way Galaxy, that galaxy to other galaxies, and so on.

How small do maps get? The map of an office desk could include the middle drawer, the map of which could include a pencil rack, the map of which could include a particular pencil, and so on. Certain technologies like cell biology and particle physics require additional levels, along with tools and units of measurement.

To return to the original problem:



By constructing the map hierarchy, I know that I have to solve the more global problem of getting from Dunham Lab to HGS. I consult my map of the Yale Campus and figure out a route.

There are two kinds of journey problems, inter-map and intra-map. To solve an inter-map problem, we use the map hierarchy, finding the lowest-level map which contains both the origin and the destination, and then solve the intra-map problem (in this case, Dunham to HGS). I believe that part of the solution of the intra-map problem depends on our visual facilities which, for example, guide the search for connected regions; if you think about getting from Boston to Chicago, how is it that you know not to go east from Boston? How do you remember direction? How do you know that San Francisco is farther from Boston than Chicago is? For the intra-map problem, I'm assuming that our visual faculties produce, in effect, a list of connected regions which are at the same level as the origin and destination regions. The route in our example is: Dunham, to Hillhouse Avenue, to Grove Street, to York Street, to HGS. TALE-SPIN does not simulate vision, but it computes this list by blind search.

Now we need to solve the problem of getting from one region to an adjacent one. I call the points of intersection doors, since the relationship of a street corner to a street is much like that of a door to a room: it's a way of getting from one region to the next.

Every door has two conceptual images, one for each region to which it is attached. People often use different terms to describe a door: the door to Room 331, the door to the hall. They know that doors are not symmetric: some doors are locked from the outside. Knowledge about a door may be incomplete: we don't always know where a door leads.



Sometimes people don't even realize that two doors are actually the same: some street intersections look different when approached along different streets.

In the current implementation, every door has two names, one for "inside" and one for "outside." Part of the map of the third floor might look like this:

(1 FLOOR3 (2 6))

(2 ROOM331 (3 4 5))

(3 DESK1 ( ))

(4 BLACKBOARD1 ( ))

(5 DOOR1 ( ) (EQV (7 1)))

(6 HALL3 (7))

(7 DOOR2 ( ) (EQV (5 1)))

This says that the third floor (FLOOR3) has two regions, Room 331 and the hall. Room 331 contains a desk, a blackboard, and a door which is marked as being "equivalent" to a door in the hall.

With the knowledge of doors connecting adjacent regions, the intra-map problem is solved by getting from the office desk to the door to Hillhouse Avenue (inter-map), getting from there to the door to (corner of) Grove Street, the door to York Street, the door to HGS, and from there to the home desk, all of which are separate journey problems. solved in turn.

The first sub-problem, for example, is getting from the office desk to the Hillhouse door. The map hierarchy tells us that this requires getting from the third floor of Dunham to the first, which might be done by going from Room 331 to the third floor hall, to the elevator, to the first floor hall, to the Hillhouse door. Eventually this produces subproblems of getting from the office desk to the office door: intra-map journeys at the bottom level are solved by actual movement, in this case, by walking. In fact, this route-finding algorithm reduces all journey problems to a series of bottom-level intra-map problems.

The theory of maps is intended to describe how people picture the spatial world. There are several related problems yet to be solved. Consider these two:

How do maps vary from person to person? Simple factual information alters maps, as in the Utah example. So do methods of transportation. A pedestrian's map is very different from a bus driver's.

How do people choose between alternate routes? The map theory doesn't say anything about that, only what routes are possible. People often choose routes which are in some sense optimal: least distance (measured along routes), least cost, least time, most familiar, least familiar (and therefore most interesting). The optimality evaluation might even be done on each piece of the route, as it's being constructed. Choice of route, then, is just another part of the goal mechanism (see Chapter 5), just like choosing planboxes in a delta-act. Although the problem domain is very different -- "A is connected to B"

is nothing like "John trusts Mary" -- it is fully integrated with the everyday domains: a precondition for taking this certain route (map domain) is that you have enough money for tolls (DELTA-CONT); a side-effect of taking that certain route (a plane, say) is that you can sleep (SIGMA-REST). While a sophisticated route-choosing mechanism remains to be built, it seems that we understand what most of the components will be like.

## 9.2 BLUEPRINTS

**P**art of the process of improvising a story is the creation of characters and objects as they are needed and not before. How do we create new map locations and connect them to the rest of the map? There are two problems here: What do we need to create when a completely new object is being added to the world model, and what do we need to create when a journey takes place? The reason that a distinction is made is that I have taken the view that one creates as little as possible while improvising. In Chapter 2 there was a story about a man who lived in a house and a bird who lived in a tree. When the house was created, we also created a valley for the house to be in, and a meadow for the tree to be in. But we did not create any physical connection between the valley and the meadow. That happens only when a journey between the valley and the meadow takes place. We did not specify that the valley and meadow were adjacent, and in general, such things can be arbitrarily

far apart. Since we don't need the connection until the journey happens, and since we have no particular reason to specify all the things between the valley and the meadow, we create as little as possible.

#### 9.2.1 WHAT'S AN OFFICE BUILDING, REALLY?

**W**hat do we start with? There are two reasonable choices. We could begin with a pre-existing world with lots of caves, houses, trees, and so on, and try to make do with that, hoping that when the reader asks for a new house (by asking for a new person, for example), that there will be an available house, one unused by any previous character. On the other hand, we could begin with nothing but an abstract model of what things are in the world, and use the model to create specific items as we need them. On the grounds that it is more flexible and more consistent with my ideas about story improvisation, I have implemented the second method, the "dynamic storage allocation" scheme, not the "fixed storage" scheme. I call the model a "blueprint." Maps are instantiations of blueprints.

What properties should the model have? It should contain all the kinds of information we need in a map system: containment, adjacency, and connection.

### 9.2.2 DIDN'T I JUST WALK OUT OF HERE?

**B**lueprints should also be minimal. Suppose our idea of office buildings says that on any given floor, there are three different kinds of rooms, one of which might be a broom closet, and that a floor can contain an arbitrary number of broom closets. The blueprint should contain exactly one broom closet, which the program should use whenever it adds a broom closet to the map.

There is a subtle problem associated with blueprint models. Suppose the blueprint for a room says that it can have two kinds of doors, doors to halls, and doors to other rooms. What other room? In the blueprint, there is exactly one room, if we insist on minimality, and we don't want to suggest that the door leads to the same room, that you could walk out of a room only to find yourself back in it. To solve this, we could include two rooms and say that the doors connect them, but their blueprints look exactly alike. The key to the problem is to identify the map which is the same, submaps of which are connected by this door. That is, we need to know whether this door connects a room to itself, or different rooms on the same floor, or different rooms on different floors of the same building, or different rooms on different floors of different buildings on the same street, and so on. In this case, we want the door to connect different rooms on the same floor, so part of the connection information for the door is the floor, the map which is "the same." This problem occurs with all self-adjacent maps.

Street corners connect streets to streets, and so on.

### 9.2.3 WHAT DO YOU MEAN, DO I WANT A NEW MOUNTAIN?

**M**ap creation is a bottom-up procedure, so that whenever a map is created, we need to create the super-map, the map in which it is embedded, unless that already exists. When is that? There are three cases when we will know what super-map is to be used.

1. If the super-map is the highest-level map possible, we use it. In the example in Chapter 2, the highest map is called THE-WORLD, and it contains mountains, valleys, and meadows, so when we create a meadow, we know what super-map it's in.
2. Some maps have "required" submaps. It's hard to imagine a building which doesn't have a ground floor, so if we've been asked to create a building, we create it and all its super-maps, and then go create a ground floor. During the course of creating the ground floor, we know we have to create its super-map, a building, but now we also know which building to use.
3. The most important case is when we create a route between two maps. The map hierarchy tells us what the doors (connections)

are between the two maps, so we create those doors. Eventually, during the course of creating successive super-maps, we'll get back to the level of the original maps we were creating the route between, and we know to use those as the super-maps, rather than creating new ones.

If none of the three cases is true, we check whether there are any maps already in existence of the same type as the one we want. If not, we just create a new one. If there is, we are left with the question: Use it or not? For example, suppose we're creating two bears. Bears live in caves, and caves are in mountains. Suppose we've created the first cave, a mountain to put it in, and the second cave. Where do we put that second cave, in the other mountain or in a new one? The program has no preference, which is to say that Context, the Great Arbiter, has not said what the answer should or shouldn't be. So the program types a message on the terminal:

IN CREATING A MOUNTAIN, WE CAN MAKE UP A NEW ONE  
OR USE AN OLD ONE. DO YOU WANT TO USE ANY OF THESE?

1: \*MOUNTAIN\*0

-- DECIDE (yes or no):

\*MOUNTAIN\*0 is the program's name for the mountain which contains the first cave. If you answer yes, the second cave will be put in the same mountain as the first cave. If you answer no, a brand new mountain will be created.



#### 9.2.4 WHAT DO YOU MEAN, WHAT KIND OF NEST DO I WANT?

**I**n discussing "minimal blueprints," we mentioned the problem of self-adjacency that arose with identical blueprints within the same super-blueprint -- in that example, rooms on the floor of an office building. There's also the problem of identical blueprints within different super-blueprints. Meadows are not valleys, but trees in meadows are just like trees in valleys. Bird nests, for instance, are found in trees. Suppose we've been asked to create a bird. Part of the bird's environment is his nest, but what kind of tree should we put it in, the trees that are in meadows or the trees that are in valleys? Again, the program has no preference, so it types:

WHAT KIND OF NEST WOULD YOU LIKE?

CHOOSE ONE OF THE FOLLOWING BY TYPING THE CORRESPONDING NUMBER

1: MEADOW            2: VALLEY

I am told that these questions are annoying, that the readers have no more preference than the program does. So there is a way to turn them off and have the program make the decision at random. But what kind of nest should you want? What would a reason be that could answer that question? It has to do with the future events in the story. That is, if you're planning to have it make a difference later on, then you know how to answer. But long-range planning is the antithesis of improvisation, so in "bottom-up" storytelling (see section 7.3), a random choice is acceptable. In "top-down" mode, however, we may have a

reason for choosing one location over another. If one of the problems in the story is going to be how Joe Bear has great difficulties in reaching Irving Bird, it makes sense to put Irving far away from Joe initially, and that can be determined by consulting the partially constructed map of the world and the blueprint: How far away can caves and nests be? Where is Joe's cave?

If the domain of interest doesn't include this DELTA-PROX, then we're obliged to make the route simple. If the reader thinks it's difficult, he will think the story is aiming towards a different point than the one the storyteller intends.

## CHAPTER 10

### SCHEDULING EVENTS IN A STORYTELLER

**S**imulating the world means simulating independent (parallel) processors. There are two typical approaches to this. The first approach is to update a clock at small regular intervals and then check each process to see what it's now supposed to be doing. The second approach, described by Gary Hendrix [13], is to let each process tell a supervisor when it should next be checked for activity. TALE-SPIN uses neither approach.

In the early versions of TALE-SPIN, there was only one character who did any planning. Other characters could respond to direct questions from him, but they were not self-motivated. Since the main character was being controlled by a top-down goal-oriented process, the problem of parallel control didn't exist. The control structure could be described as a stack mechanism.

But now characters respond independently. Each character has his own goal structure, which is changed when he learns new things and formulates his own plans. We get around the problem of parallel processes by taking advantage of the linear nature of stories: since the storyteller is only saying one thing at a time, he only needs to worry about one thing at a time. If John is the focus (as opposed to main character), and he asks Mary a question, she now becomes the focus. We don't worry about John until Mary replies to his question. In the stories TALE-SPIN now produces, people wait for replies, so we're not missing anything if we ignore John while Mary is the focus. It's still one character at a time, just not the same character all the time, so the control structure can still be described as a stack mechanism.

The focus changes from character to character when new goals are asserted. Sometimes this is done directly. Spoken requests are the most obvious case of a direct change of focus. After Joe Bird asks Irving Bird to do something, we expect to hear about Irving Bird. With the implementation of the ability of characters to notice things in a new environment, however, new goals were indirectly asserted. If Henry Fox happens to walk up to Bill Crow who's holding a piece of Henry's favorite brand of cheese, Henry automatically acquires the goal of satisfying his hunger (unless he has eaten recently). No verbal communication or goal-directed planning is necessary.

Promises present special problems. A promise is a context in which special inferences can be made. "John looked at the clock and saw that it was midnight." While there are inferences John might now be able to make -- he might decide to go to bed -- consider what inferences John would make had he promised to call Mary at midnight. We want to make the inference that John is going to call Mary, but how do we do that unless we check, every time John finds out what time it is, whether John has promised to do something at that time? The answer is, of course, to use demons, special tests which appear and disappear as needed. If John has promised to do something at midnight, then we do check whether it's midnight every time he finds out what time it is, until it's midnight and he carries out the promise, at which point we remove the demon so that the next time John sees what time it is, no tests are performed.

In the story in Chapter 2, Wilma Canary promises to tell Sam Adams where some berries are if he brings her a worm. He does, but she simply eats the worm and calls him stupid. The relevant point here is not the trick, but her reaction to receiving the worm. Not only did she eat the worm, which is always expected when someone is hungry and is given food, but she did something else as well. That was done with a demon: when she made the promise, a demon was created, to be triggered when Wilma got a worm from Sam.

The arrival of demons signals the eventual disappearance of the stack mechanism. Why? Because using a stack to do promises can lead to a deadlock. A detailed example shows how.

reason for choosing one location over another. If one of the problems in the story is going to be how Joe Bear has great difficulties in reaching Irving Bird, it makes sense to put Irving far away from Joe initially, and that can be determined by consulting the partially constructed map of the world and the blueprint: How far away can caves and nests be? Where is Joe's cave?

If the domain of interest doesn't include this DELTA-PROX, then we're obliged to make the route simple. If the reader thinks it's difficult, he will think the story is aiming towards a different point than the one the storyteller intends.

## CHAPTER 10

### SCHEDULING EVENTS IN A STORYTELLER

**S**imulating the world means simulating independent (parallel) processors. There are two typical approaches to this. The first approach is to update a clock at small regular intervals and then check each process to see what it's now supposed to be doing. The second approach, described by Gary Hendrix [13], is to let each process tell a supervisor when it should next be checked for activity. TALE-SPIN uses neither approach.

In the early versions of TALE-SPIN, there was only one character who did any planning. Other characters could respond to direct questions from him, but they were not self-motivated. Since the main character was being controlled by a *top-down goal-oriented process*, the problem of parallel control didn't exist. The control structure could be described as a stack mechanism.



Here is an imaginary procedure for John to find out from Mary what time it is:

(John) Begin  
    Ask Mary what time it is;  
    If you now know the answer, succeed; Else, fail;  
End;

How could John possibly know the answer right after asking the question? Easy. As soon as he asks the question, Mary acquires a new goal of answering him. The top-level control of TALE-SPIN works on new goals as soon as they appear. The focus now shifts to Mary. Suppose she already knows the answer. Then she tells him the answer. The inference from telling him is that he now knows the answer. *Nothing else happens* -- John makes no special inferences, Mary has no more goals -- so we're finished with that part, and we "return" to the point at which John asked the question, in the procedure above. In step 2, we test whether John knows the answer. Surprise, surprise, he does, so all is well. This mechanism works for requests, but not for promises. If we use the same mechanism, we might imagine the following procedures:

(John) Begin  
    Tell Mary that if she tells you what time it is,  
        then you'll bring her an apple.  
    If you now know the answer, bring Mary an apple.  
End

(Mary) Begin  
    Tell John what time it is.  
    If he doesn't bring you an apple, get very angry.  
End.

In the stack model, as soon as Mary tells John what time it is, she immediately expects to have an apple, but he's not going to do that

until we "return" to his level. So, in a sense, she's waiting for him and he's waiting for her -- deadlock.

The solution is to create a demon for him, saying that as soon as he finds out from her what time it is, then he'll get her an apple. If we do that, we can even keep her procedure the way it is. TALE-SPIN, in fact, uses both the stack and demon mechanisms.

But even the demons don't allow us to simulate simultaneous actions. The example above still has one character acting at a time. Would we ever need more? Here's a TALE-SPIN story:

Once upon a time there were two bears named Jack and Joe, and a bee named Sam. Jack was very friendly with Sam but very competitive with Joe, who was a dishonest bear. One day, Jack was hungry. He knew that Sam Bee had some honey and that he might be able to persuade Sam to give him some. He walked from his cave, down the mountain trail, across the valley, over the bridge, to the oak tree where Sam Bee lived. He asked Sam for some honey. Sam gave him some. Joe Bear walked over to the oak tree and saw Jack Bear holding the honey. He thought that he might get the honey if Jack put it down, so he told him that he didn't think Jack could run very fast. Jack was challenged by this and decided to run fast. He put down the honey and ran over the bridge and across the valley. Joe picked up the honey and went home.

What would happen if Jack Bear were a little smarter and figured out that he'd been tricked? What should happen? He should get mad and come back, looking for Joe. And here's the problem: If he does that, Joe will still be there, because goals are pursued as soon as they arise. While Joe certainly has the goal of picking up the honey and going home, the simulator wouldn't get around to working on that goal before Jack returns, because the focus never leaves Jack -- he keeps setting up new goals for himself, and new goals get first priority.

What we really want is for Joe to be picking up the honey and heading home at the same time that Jack is figuring out that he's been duped. The way to simulate that is easy: we change the top-level loop. Instead of simply achieving the most recent goal, it should examine the most recent goal of each character, and choose the oldest of those, not the newest, to work on next. Since all events in the story, including goals, have a specific time of occurrence as part of their representation, we can compare them to find the oldest. But before we start worrying about how to resolve ties in such a comparison, there's a more serious issue: the fable and the sujet.

Russian folklorists make a distinction, in French of course, between a story as it happens and a story as it is told. The former is called the fable, the latter, the sujet. Flashback is a common technique for altering the order in which events are told. "Meanwhile, back at the ranch, ..." is a common phrase indicating to the reader that the clock is being reset to some previous time, so that we can now hear about what was going on at the ranch at the same time as the part of the story we've just heard. Why do this? Because it's easier to follow one character at a time and reset the clock when we switch characters than it is to follow the clock and switch characters all the time. That's a rule of stories: people prefer that sujet to the corresponding fable. Consider how a human might have told the ending of our altered story:

When Jack got to the other side of the valley, he realized that he'd been tricked into giving up the honey. So he ran back to the oak tree, but Joe had already picked up the honey and gone home.

We stay with Jack until he is supposed to see Joe, at which point we

consider whether Joe is still there. If not, we say that he left, etc.

The alternative is to describe the fable:

At 3:01, Jack reached the valley. At 3:02, Joe picked up the honey. At 3:03, Jack realized he'd been tricked, and he headed back for the oak tree. At 3:04, Joe headed for his cave. At 3:05, Jack reached the oak tree. At 3:06, Joe Bear reached his cave....

That style is used for describing the exciting moments at the end of novels -- it seems to be one of Arthur Hailey's favorite devices -- but it's hardly appropriate for an entire story, and that's the problem: the proposed changes to the top-level would fix TALE-SPIN so that simultaneous events would be handled correctly, but the resulting fable could not be expressed directly.

A now-obvious project for future work is to define a theory of the sujet, to define the rules for altering the temporal order of the events in a story. Again, we see how the typical computer-system approach to solving certain problems -- in this case, control structure -- corresponds badly to the approach used by humans, and how we have to discover new and more elegant explanations.

## CHAPTER 11

### DETAILS, DETAILS

#### 11.1 INTRODUCTION

**F**or the benefit of those who wish to see the innards of TALE-SPIN, I present this chapter. I'm going to follow the generation of a story from start to finish, giving as much detail as I can. I'm also going to describe the related sources of information, following, in effect, all the alternate paths in the maze as well as the correct one. Be forewarned that you may lose your bearings. Only computers are immune to such headaches.

Here's the story in brief: Arthur Bear is hungry. He has this devious friend, George Bird, who lies to him by telling him that some honey exists when it really doesn't. Arthur tries very hard to get the honey anyway, eventually failing.

I chose a "failure" story because it shows more of the planning alternatives than a "success" story does.

### 11.2 A VERY DETAILED EXAMPLE

\*\*\*\*\* WELCOME TO TALE-SPIN \*\*\*\*\*

CHOOSE ANY OF THE FOLLOWING CHARACTERS FOR THE STORY:

(BEAR BEE BOY GIRL FOX BIRD ANT HEN LION DOG WOLF MOUSE  
CAT GOAT)

\*(BEAR BIRD)

We want to create a bear and a bird. Associated with each generic name is a "picture procedure" which specifies the details for creating the item. The picture procedure for creating a bear actually creates both a bear and a cave, marks the cave as being the bear's home, and lets the Arthur and the storyteller know that the bear is in the cave.

### 11.3 DIGRESSION: MINIMAL ENVIRONMENT

**W**hat is a bear? One way to answer that would be to consult a zookeeper, a forest ranger, or an animal psychologist. But another way is to ask the man on the street. In fact, modeling this more common knowledge is our real goal. What does the man on the street know about bears? For one thing, he knows that you can find out detailed information about bears by asking a zookeeper, a forest ranger, or an

animal psychologist. That is, the source of detailed information about an object is itself information about the object.

Physical characteristics are common knowledge. "We saw a grizzly bear yesterday. Its claws were six inches long." No problem. "We saw a goldfish yesterday. Its claws were six inches long." Impossible. "We saw a grizzly bear yesterday. Its claws were sixty inches long." Impossible.

Of all the information we know about an object, we need only put some of it into a computer program, namely, that which is relevant to the story, that which will be needed. We can classify information in three ways: (1) immediately necessary, (2) never necessary, and (3) possibly necessary. For example, as soon as we create a character, we need a name, because it is customary to use proper names. Totally unnecessary information is that which we know in advance will alter nothing, information which will not matter at all. TALE-SPIN doesn't assign ages to its characters because there's no test which says "If X is less than twenty years old, then do such-and-such, else do such-and-such." This is equivalent to saying that the concept of age is unknown to TALE-SPIN. The day we decide to put in a need for such a test, we'll add age to the characters.

Possibly necessary information is that which may be used at some point, but might not. These details are established as they are needed. Physical locations are in this class.



The process of creation may or may not be deterministic. We need a name for every character, but the particular choice is unimportant. Such non-deterministic choices are either made at random or given to the reader to decide.

#### 11.4 CREATING A BEAR

**S**o when we create a bear, we also create a cave which will be the bear's home. This is the bear's minimal environment. We test whether the item being created has the property SENTIENT. Part of the initial world knowledge that TALE-SPIN has is that all characters, including bears, are sentient. Other items, such as trees, are not. Next we test whether the item has a gender already attached to it, as it would if we had been asked to create either a boy or a girl. But "bear" can be either male or female, so we choose a gender at random, with 50-50 odds, male-female.

Next we choose a name for the character, choosing at random from a list of boys' names. Each character has several kinds of names. The program uses exactly one; all the others are for the sake of the generator. Our bear gets the internal name \*ARTHURBEAR\*; his "proper name" is ARTHUR BEAR; and his "first name" is ARTHUR. This name-giving procedure also knows whether to elide the parts of the proper name, as it would in the case of berries. The list of names for berries is:

CRAN BLUE RASP STRAW BLACK BOYSEN



The proper name will come out as BLUEBERRIES, not BLUE BERRIES.

#### 11.5 DIGRESSION: RANDOM CHOICES VS. THEORY

A brief word about random choices is called for. When a choice has to be made and there's no theory around to make it, TALE-SPIN chooses at random. Choosing the first name of a character is an example. The storyteller has no reason to prefer "John" over "Arthur" or any other name in the list. There's nothing which the storyteller has in mind which fixes the choice of first name, so it chooses at random. If Arthur Bear is hungry and doesn't know where any food is, he'll choose a food at random and start looking for it. Obviously, that's a very simple strategy. A "theory" about choosing a food would require a goal calculus -- what food is nearby? Cheap? Available? But the point is that not even random choices are free of side-effects. It makes a difference which food Arthur Bear picks, since he'll look in different places for different foods.

The part of TALE-SPIN which has the most theory behind it is the simulation of planful behavior. As long as there are goals, the program knows a great deal about what a character should do. Choosing "what to do next" at random means that nobody is motivated to do anything in particular.

#### 11.6 CREATING A BEAR, CONTINUED

**N**ow that we have the name \*ARTHURBEAR\*, we add some properties: Arthur is of type \*BEAR\*; the generator treats him as it would a human (using personal pronouns and so on); his internal name, \*ARTHURBEAR\*, is added to !PERSONAE, the list of characters; his proper name, ARTHUR BEAR, is added to !CAST, a parallel list. His height is chosen at random, anywhere between 60 and 80 inches, and his weight, anywhere between 4000 and 6000 ounces. (For convenience, the same units of measurement are used for all characters.)

Now we have to create a cave, or, more properly, a \*CAVE\*. Caves are non-sentient generics, so we use a different procedure than we did before. Another part of TALE-SPIN's initial information is an abstract representation of the physical world, called a blueprint. (Blueprints and their concrete counterparts, called maps, are discussed in detail in Chapter 9.) We test whether there is a blueprint for the item we're creating. There is indeed a blueprint for \*CAVE\*. It's part of the blueprint for \*MOUNTAIN\* which is part of the blueprint for \*WORLD\*. If there's more than one blueprint, then you get to choose one. (See section 9.2.4, "What do you mean, what kind of nest do I want?") There is only one blueprint for \*CAVE\*, so that's not a problem. Next, we test whether there are any caves already in the world -- you might want to use one of them instead of making up a new one. (See section 9.2.3, "What do you mean, do I want a new mountain?") There aren't any caves in the world yet. In fact, the only thing that is in the world so far


is the highest-level physical object, called THE-WORLD. So we make up a new cave and give it the internal name \*CAVE\*0. Generic physical objects don't get proper names as characters do. Instead, they get "English names," in this case, the name "cave." (The generator I now use, described in Chapter 12, uses only English names. The standard generator, a descendant of Goldman's BABEL, uses names in several languages, including Chinese and Russian.) \*CAVE\*0 is given some other properties: it's a cave, it's a \*PHYSOBJ\* (physical object), it's singular, it's not open ground (as a meadow is, for instance), it's not physically connected to anything yet.

Creation of physical objects is done in bottom-up fashion. Now that we have a cave, where is it? Well, the blueprint tells us that caves are in mountains, so we now start the procedure over and create a mountain. It gets the internal name \*MOUNTAIN\*0, the English name "mountain," and so on. Eventually, we ask the same question, where are mountains? The blueprint specifies that mountains are part of THE-WORLD. Since we only permit one physical world, the process stops there. We add \*MOUNTAIN\*0 to the (previously empty) list of things in THE-WORLD, and we add \*CAVE\*0 to the list of things in \*MOUNTAIN\*0.

We're done, having created \*ARTHURBEAR\* and \*CAVE\*0 directly, \*MOUNTAIN\*0 indirectly. \*CAVE\*0 is marked as being \*ARTHURBEAR\*'s home. The fact that Arthur is in his cave is now added to Arthur's memory and to the storyteller's memory. (The storyteller's memory represents what's true. See Chapter 6 for details.)

And that's how Arthur Bear is created. Now we want to create a bird. Creating the bird is much like creating the bear. We need a nest instead of a cave. Omitting the details, this bird's name came out to be \*GEORGE BIRD\*.

#### 11.7 CREATING PHYSICAL LOCATIONS

reating the nest is a little harder. There are two blueprints for nests, one for the nests in trees in valleys, the other for nests in trees in meadows. When this story was actually produced, it did not ask the reader which we wanted, but decided at random in favor of the meadow variety. (The decision to ask is an option we can set at the very beginning.) So we create \*NEST\*0. Since nests are in trees, we need to create a tree as well. Trees have proper names, according to the kind of tree. The name-giver chooses at random from the list:

ELM OAK MAPLE FIR DOGWOOD APPLE CHERRY REDWOOD ASH

This time it produced \*MAPLE TREE\* (proper name: MAPLE TREE). Trees may or may not have nests -- there is a separate picture procedure for creating trees without nests in them -- but all trees are connected to the ground, and TALE-SPIN regards the bottom of the tree as a separate part of the tree, so after the tree is created, we create some "ground" around the tree. Its internal name is \*GROUND\*0. Its English name is "GROUND BY THE MAPLE TREE." That's a little long-winded; people generally say "John walked to the maple tree," not "John walked to the ground by the maple tree," but they mean the latter. For the sake of

being precise, then, TALE-SPIN is a little long-winded.

The tree is in a meadow, and there aren't any meadows yet, so we conjure up \*MEADOW\*0. It becomes part of THE-WORLD, \*MAPLETREE\* becomes part of \*MEADOW\*0, \*NEST\*0 and \*GROUND\*0 become part of \*MAPLETREE\*. Finally, we mark the tree as being George's home, and we let the storyteller and George know that George is in the nest.

Part of the knowledge that all the initial characters have is the "true" location of all the other characters. The "true" location is where they really are. When the story begins, it will be possible for the characters to acquire all sorts of misinformation, and in fact, whenever any character goes to visit any other character, the program has him go to the place where he thinks the other character is, which may or not be correct. But we'll be nice for now, telling Arthur Bear that George Bird is in the nest in the maple tree, and telling George Bird that Arthur Bear is in his cave.

Arthur and George are also made known to each other by name: Each character has a list of "acquaintances," other characters that he's aware of. No particular relationship is implied by being an acquaintance.

Having created the characters, the program now asks us if we want anything else in the world.

CHOOSE ANY OF THE FOLLOWING MISCELLANEOUS ITEMS:  
(BERRIES FLOWER RIVER WORM)  
\*(WORM)

Worms are not considered to be sentient beings. (In an early version of TALE-SPIN they were, but it seemed awkward for Joe Bear to fetch "Sam Worm" to feed to Irving Bird. "A worm" was much better.) \*WORM\* is therefore not a sentient generic, but nor is there a blueprint for it, so we simply create \*WORM\*0 and leave it at that. Worms are found in the ground, so we need to create a new \*GROUND\* and a new \*MEADOW\* to put the \*GROUND\* in. There is already a meadow in existence, but we used the option that disables re-use of old locations, so we create \*MEADOW\*1 and \*GROUND\*1. The picture procedure specifies that the worm is in the ground. But now the program asks

WHO KNOWS ABOUT THE WORM?

1: GEORGE BIRD      2: ARTHUR BEAR

\*2

Since the worm isn't a character, it doesn't have any knowledge. The storyteller knows that the worm is in the ground, but none of the characters do. If they don't, there's no way that the worm could ever be part of the story, since the characters rely on their knowledge to do everything.

Since we specified that Arthur Bear knows about the worm, the notion "Arthur Bear knows that the worm is in the ground" is now added to memory, along with all its inferences. This will enable Arthur to find the worm later, for instance. Inferences from knowledge states ("John knows that ...") are called reactions, and they depend on the particular information and on the person who has the knowledge. In this case, Arthur Bear has no particular reaction to knowing that there's a

worm somewhere. Had it been not a worm but a blueberry bush instead, he would have reacted by planning to eat the berries, since berries are among the things bears eat.

#### 11.8 THE FIRST PROBLEM

*If*

inally, we are ready to begin the story. The program types:

THIS IS A STORY ABOUT ...

1: GEORGE BIRD      2: ARTHUR BEAR      \*2

HIS PROBLEM IS THAT HE IS ...

1: HUNGRY      2: TIRED      3: THIRSTY      4: HORNY      \*1

The four "problems" are called *s'gma*-states and correspond to physical needs. The program generates the representation for "Arthur knows that he's hungry" and asserts it. The notion "Arthur is hungry" is stored in Arthur's memory, then its inferences are computed and also asserted. There is a reaction-inference procedure for each of the states and acts in CD. The one for the state \*HUNGER\* asks whether it's a positive or negative state of hunger -- there's no reaction to "John isn't hungry." It also asks whether the degree of hunger is negative -- there's no reaction to "John is full," whose representation differs from that of "John is starving" only in degree. Both tests succeed in this case. The reaction, then, is that Arthur intends not to be hungry because he wants to preserve his health.

Arthur is the one who has reacted to this, so the inference from "Arthur knows that he is hungry" is, literally, that "Arthur knows that



Arthur intends not to be hungry." When the assertion-maker sees anything of the form "X knows that X intends to do Y," it invokes the planning structure appropriate to Y; that is, the goal Y is now going to be pursued.

It is necessary to restrict that to "X knows that X intends to do Y." Otherwise, if we activated the planning structure whenever we came across "X intends to do Y," we couldn't distinguish between "John thinks that Arthur intends not to be hungry" and "Arthur thinks that Arthur intends not to be hungry." We would make the first of those inferences if John found out that Arthur was hungry, but it's only legitimate to say that Arthur is really going to do something about it when he knows.

#### 11.9 SIGMA-HUNGER

**A**ssociated with the state \*HUNGER\* is S-HUNGER (SIGMA-HUNGER), which is the name of the procedure for satisfying hunger. S-HUNGER's first task is to specify some food for Arthur to eat. Part of TALE-SPIN's input data is the fact that bears eat honey and berries. (Bees eat flowers, birds eat worms and cheese, foxes and mice eat cheese, and so on.) The list of bear-foods is used as S-HUNGER asks memory whether Arthur knows that he owns some honey, knows where some honey is, knows that he owns some berries, or knows where some berries are. At this point, all Arthur knows is that he's in the cave, that he's hungry, that there's a worm out in some patch of ground, and that



George Bird is in his nest. So S-HUNGER selects a bear-food at random, in this case, honey, and invokes DCONT (DELTA-CONTROL).

#### 11.10 DELTA-CONTROL

**D**CONT forms the goal "Arthur has some honey" and checks to see whether the goal is already true, in which case it would succeed trivially. (Although this particular test was performed before, back in S-HUNGER, we repeat it here. DCONT is also used in situations where there is less specific knowledge.) Otherwise, it goes on to check that this goal isn't already part of Arthur's goal structure. It isn't -- the goal structure was empty -- so it is added. Had it been there, DCONT would have failed, because it would mean that getting honey was either a current goal, in which case it makes no sense to pursue it as a subgoal, or a past goal that was unachievable, in which case it won't do any good to try it again.

#### 11.11 DELTA-KNOW

**T**he first precondition in DCONT is to find out, via DKNOW (DELTA-KNOW), where some honey is. DKNOW forms the goal "Arthur knows where some honey is" and checks whether it's already true. It isn't -- Arthur doesn't know where any honey is. DKNOW then checks this goal against the goal structure. It isn't there, so it's added.

The first planbox (method) to be tried asks: Is there a standard method that could be used to answer this question? This is intended to allow the use of scripts [27,28,30], such as looking at a wristwatch to find out the time. It's something you do automatically, without having to do much thinking. A plan for finding out the time might include taking measurements on the position of the sun, which makes sense only if all else fails. Looking up things in phone books and dictionaries are other examples of DKNOW scripts.

But there isn't a script for finding out where some honey is, so we try the next planbox, which asks: Is this a "general information" question? "Where is ...?", "Who has ...?", and "Who knows ...?" are the general-information questions presently used by DKNOW. Since this question matches the first type, we can use the planbox. It says to PERSUADE a friend to tell you the answer. X is a "friend" of Y when X thinks that X relates to Y with a positive amount of affection. All relationships are expressed as states in CD and stored that way in memory.

#### 11.12 RELATIONSHIPS

**T**esting memory for relationship states is done by a special procedure called REL which will add the relationship to memory if it isn't there. That is, we assume that everyone relates to his acquaintances with some degree of affection, even if it's zero. Normal



AD-A031 625

YALE UNIV NEW HAVEN CONN DEPT OF COMPUTER SCIENCE

F/G 9/2

THE METANOVEL: WRITING STORIES BY COMPUTER. (U)

N00014-75-C-1111

UNCLASSIFIED

RR-74

NL

2 OF 2

AD  
A031 625

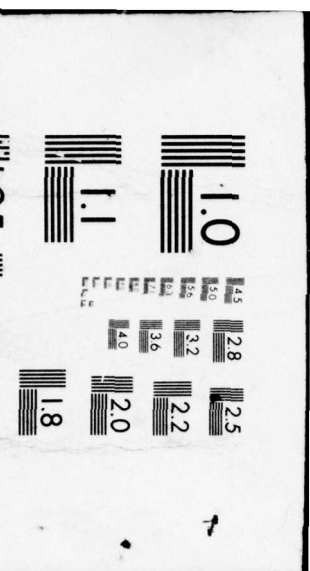


END

DATE  
FILMED  
12-76



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A



memory tests simply look for a fact and fail if they don't find it. If REL doesn't find a relationship between two characters, it asks the reader. In this case, it types the following question:

DOES ARTHUR BEAR THINK THAT GEORGE BIRD LIKES HIM?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL

\*1

It's important to distinguish the directions of a relationship (who likes whom, for example) and to specify who believes that the relationship exists. For example, there are four possible ways of expressing the "affection" relationships between Arthur and George.

1. Does Arthur think that George likes him? This would be asked when Arthur is considering whether he can ask George to do him a favor.
2. Does Arthur think that he likes George? (This is equivalent to: Does Arthur like George?) This is asked when Arthur is considering honoring a request George has made.
3. Does George think that Arthur likes him? This would be asked when George is considering whether he can ask Arthur to do him a favor.
4. Does George think that he likes Arthur? (This is equivalent to: Does George like Arthur?) This is asked when George is considering honoring a request Arthur has made.

The difference between 1 and 4 is that 1 is stored in Arthur's memory, 4

in George's; likewise for 2 and 3. Of course, there can be a third party involved: Does Mary think that Tom likes Wilma? All the relationships -- competition, dominance, familiarity, affection, trust, deceit, and indebtedness -- are handled this way.

The social preconditions in the planboxes always specify a range of acceptable values. When considering a request, for instance, an "affection" value between +6 and +10 will prompt an automatic acceptance, whereas a value between -10 and -6 will prompt an automatic refusal. The REL function can process arbitrary logical combinations: Is "affection" between 1 and 3 and "trust" between -1 and 2, or is "indebtedness" between 4 and 10? REL stops asking questions as soon as it knows enough to get the final answer, so if you ask about "A and B and C," and A is true but B is false, it won't ask about C.

There is an analogous function for the personality states: kindness, vanity, honesty, and intelligence.

In order to find a friend to persuade, then, DKNOW goes through the list of Arthur's acquaintances until it finds someone who thinks of himself as Arthur's friend. According to the answer we gave, Arthur thinks that George likes him a lot, so George is the first candidate for persuasion.



# 11.13 PERSUADE

The function PERSUADE takes three arguments: the person doing the persuasion (Arthur), the thing he wants done (that George tell him where some honey is), and his goal behind all this (that he know where some honey is). Usually the goal is one of the consequences of the act, but not always:

Tom and Mary weren't speaking to each other. At dinner, Tom asked his daughter, "Heather, would you please ask your mother to pass the salt?"

Tom's goal is to get the salt, but that may or may not happen even if Heather does what she is asked.

The first planbox in PERSUADE is to ask, but there are preconditions on the relationship which have to be considered before asking. If Arthur feels deceptive toward George (in the range 1 to 10) or competitive (3 to 10) or hostile (affection -10 to -5), or if Arthur thinks that George feels hostile toward him (-10 to -1) or deceptive (1 to 10), then asking is ruled out. (The particular numbers, of course, are simply rough guesses.) The REL function produced this question/answer session:

DOES ARTHUR BEAR FEEL DECEPTIVE TOWARDS GEORGE BIRD?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

DOES ARTHUR BEAR FEEL COMPETITIVE TOWARDS GEORGE BIRD?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

DOES ARTHUR BEAR LIKE GEORGE BIRD?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*1

DOES ARTHUR BEAR THINK THAT GEORGE BIRD IS TRYING TO DECEIVE HIM?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

It did not ask whether Arthur thought that George liked him, since it already knew the answer to that question.

#### 11.14 TELL

**N**ow that George has "passed the test," Arthur can ask him where the honey is. PERSUADE calls the function TELL, which is the procedure for communicating information (as opposed to finding out information, which is what DKNOW does). The message Arthur wants to "tell" George is: Will you tell me where some honey is?

TELL has two planboxes: you tell the person yourself, or you get a friend to tell him. The precondition for the first one is that you be physically near the person, so TELL calls DPROX to get Arthur near George.

#### 11.15 DELTA-PROX

**D**PROX forms the goal "Arthur is near George." In general, DPROX handles "A wants B to be near C." The new goal is added to Arthur's goal structure. A precondition on all of DPROX's planboxes is that B be movable. The initial world knowledge specifies what kinds of things are movable: Characters (sentient creatures) are, beehives are, flowers are, trees are not, and so on. Characters are also known to be self-movable, and with each type of character is listed what part of his

body he moves in order to get himself from place to place. Bears, ants, humans, and foxes walk (\*LEGS\*); birds and bees fly (\*WINGS\*); worms wriggle (\*BODY\*). The first planbox in DPROX is for Arthur Bear to move himself to George Bear, so the precondition is that Arthur be self-movable. If A wants B to be near C, and A isn't the same as B, then a precondition is that A get near B (again using DPROX) and grab C. That isn't the case here: Arthur wants to get himself near George. Next, Arthur needs to find out where George is, so DPROX calls DKNOW again. DKNOW, as always, checks to see whether Arthur already knows where George is, and he does: George is in \*NEST\*0. The next precondition is that Arthur find out where he himself is. If he doesn't know that, he can't very well go anywhere. DKNOW is called yet again, and memory answers that Arthur is in \*CAVE\*0.

#### 11.16 HOW CLOSE IS CLOSE?

**N**ow the question is asked: Are the two locations close to each other? "Close" is good enough; "equal" isn't necessary, although it'll do. The ground around a tree is "close" to anything in the tree. One side of a door is "close" to the other side, and this applies to all "doors" (a technical term defined in Chapter 9) such as street corners and corridor intersections.

Even with the leeway permitted there, Arthur isn't close to George, so we call the travel agent, DLINK (DELTA-LINK), for a path from \*CAVE\*0

to \*NEST\*0. DLINK is an odd delta-act in that it never fails -- there is always some route between any two locations. There may be problems in the journey itself, but not in thinking up a route. TALE-SPIN's world, however, lacks transportation systems, and if they were included, knowledge gaps (timetables and so forth) could very well cause a failure. Schedules would also impose new time constraints on the goal.

#### 11.17 DELTA-LINK

**A**t this point, the map of the world includes a mountain with Arthur's cave and a meadow with George's tree. Now, however, we need to connect them. The blueprint for the world specifies that mountains do not adjoin meadows, but they are adjacent to valleys and valleys are adjacent to meadows, so we set about creating a valley, using the same procedures we did when created the mountain and the meadow.

DLINK always chooses the shortest possible path, simply because the storyteller has no reason to make journeys *deliberately* long. Instead of creating one valley which adjoins both the mountain and the meadow, we could create a valley next to the mountain, another mountain on the other side of the valley, another valley, and then connect the meadow, but what's the point? If we knew we wanted to tell a story about a difficult journey, then we could make it very complicated, but we're not in that mode, so we settle for the easiest solution.

Creating a path is more detailed than simply creating some physical

location. We need all the connections along the way. The mountain blueprint specifies that the point of intersection between a mountain and a valley is the end of a mountain pass, so we create a mountain pass. The information about caves says that they adjoin passes directly, the point of intersection being the cave opening, so we create an opening for the cave. All points of intersection have two names, one for each area to which they're attached, and the names are marked as being "equivalent." So the opening has two names, the cave entrance (its name on the outside) and the cave exit (its name on the inside). Similarly, the end of the mountain pass is known as the pass-valley border and the valley-pass border, and we also get a valley-meadow border and a meadow-valley border. All this information about connected regions and paths is now added to the representation of the actual world. Meadows and valleys are "open ground," meaning that you don't need a special route to get from one point in them to another.

So DLINK has made a route for Arthur. DPROX now calls DO-PTRANS, which is an "action module" (defined in section 3.13), to do the actual movement. DO-PTRANS puts on the final touch: Since Arthur can't fly, \*GROUND\*O, the ground by the maple tree, is as close as he can get to \*NEST\*O where George really is. DO-PTRANS now asserts that the Arthur takes the trip. It is the first act in the story. The inference from the PTRANS is that Arthur is at the ground by the maple tree. Had there been other characters back at the cave, other inferences would be made, namely, that each of them now knows that Arthur isn't at the cave any more.

The assertion-maker now computes inferences from the fact that Arthur is at the tree. The inference procedure asks for the names of all the characters "close" to the tree. It needs the real, true information, not merely what some character thinks, because the characters near the tree are going to see Arthur whether or not he expected them to be there. "Real, true" information is, of course, what the storyteller knows, and this is a good example to show why we need to keep a separate memory for the storyteller.

#### 11.18 DIGRESSION: NOTICING

**A**n important part of the simulator controls the action of independent processes, especially the action of people. This leads to a control-structure problem, a conflict between the top-down structure of goal-directed processes and the parallel structure of independent processes, discussed in Chapter 10. The difference lies in the strength of the causality of actions. If John wants to eat an apple, he can simply grab it and eat it. He's the only actor involved, the only "processor," dependent on nothing else. Independent processes, however, affect each other by triggering reactions, a less direct causality. If John wants Mary to see him, he can walk up to her (in her line of sight, presumably). He does nothing more. Her vision is triggered -- she notices him -- but that is outside John's direct control.

From "A is at location B," TALE-SPIN infers that A now knows about

all the things which are at or near B, and that all the other characters near B now know about A and anything A may be holding. The notion "near" depends upon the size of the location B relative to the size of A. If John walks into an office building, we don't expect the people in the adjacent office building to notice, but if he sits on the left end of a couch in a living room, we do expect him to be noticed by the people on the right end, and in fact, by everyone else in the room, as well.

\*\*\* End of Digression \*\*\*

(DELTA-LINK, continued)

The characters close to the tree are Arthur Bear and George Bird, so the inferences are that Arthur knows he's at the ground and that George knows that Arthur's at the ground. We assume that all the characters can see well, and we don't handle line-of-sight problems yet, which could limit these inferences. If anything or anybody near the tree is holding anything (or is being held), everyone gets to find that out -- no hiding things in pockets yet.

We now compute the inferences from the fact that Arthur Bear knows that he's at the tree. There aren't any. If he had found himself in the river, an inference would have been that we wanted to get out, lest he drown. If he had found himself in the presence of some bear-food, a satisfy-hunger goal would have resulted. If he had found himself near



some object he likes to own (\*BOY\*s like to own \*BASEBALL\*s), he would have acquired the goal of possessing it.

Similarly, there are no inferences from the fact that George knows that Arthur is at the tree, so the program control returns to DPROX, which now tests to see that the DO-PTRANS actually succeeded. The story will include an example where it failed, so I'll postpone discussion of what happens then until later.

DPROX has succeeded, and returns to TELL. (Arthur was going to ask George to tell him where some honey was.) Had DPROX failed, TELL would try its second planbox, which is to PERSUADE a friend to communicate the message.

#### 11.19 DO-MTRANS

**T**ELL now calls DO-MTRANS. The second event of the story is about to happen. Arthur asks George to tell him where some honey is. In DO-MTRANS, we test whether someone is "telling himself" something, e.g., remembering something. If so, we simply assert that it happens. Otherwise we test to see whether the person doing the communicating is under water. This is a runtime precondition which will prevent the communication from happening. Neither of those is true now, so DO-MTRANS asserts that the MTRANS occurs: Arthur asks George if he'll tell him where some honey is. The inference is that George knows that Arthur has asked him the question. Had it been a statement of fact



rather than a question, an additional inference would be that George thinks that Arthur believes what he said. There are no reactions to an MTRANS -- George doesn't react to the fact that Arthur happened to speak to him.

#### 11.20 REQUEST

**B**ack in DO-MTRANS, however, we test whether X has just asked Y to do Z. That is the case here, so DO-MTRANS calls REQUEST, a special procedure for handling favors. In deciding whether George is going to do what Arthur has asked, REQUEST tests some more parts of the relationship between the two.

DOES GEORGE BIRD LIKE ARTHUR BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

DOES GEORGE BIRD TRUST ARTHUR BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

DOES GEORGE BIRD DOMINATE ARTHUR BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

If George had disliked Arthur, or not trusted him at all, or felt that he was in a position of dominance over Arthur, then he would have refused.

DOES GEORGE BIRD FEEL INDEBTED TO ARTHUR BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

It also checks whether George likes Arthur very much -- it already knew the answer, so it didn't ask here -- or if George felt that he owed something to Arthur. If so, he would have agreed to do what Arthur

asked.

But the relationship doesn't provoke either an automatic refusal or an automatic agreement, so George thinks about what it would mean to him if he did tell Arthur where some honey was. He does this, of course, by considering the inferences from telling him. Not only the storyteller uses the inference mechanisms. Characters can, too. The inferences are that Arthur would know that George told him where some honey was and that Arthur would think that George thinks that there really is some honey there. There are no further inferences from the first one, but there may be an inference from the second. If Arthur trusts George, then he'll believe what George says. So the inference-maker asks:

DOES GEORGE BIRD THINK THAT ARTHUR BEAR TRUSTS HIM?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

"A little" is enough. The new inference is that Arthur thinks that the honey is where George said it was. An inference from that is that Arthur will plan to eat the honey.

Those are the inferences George makes, and now he considers how he feels about them, how he reacts to them. He does this by making the inferences from believing each one of them, namely:

1. George thinks that Arthur thinks that George told him where some honey is.
2. George thinks that Arthur thinks that George thinks that there really is some honey there.

3. George thinks that Arthur thinks that the honey is where George said it was.

4. George thinks that Arthur is planning to eat the honey.

But all this thinking hasn't answered George's initial purpose: He was looking for an inference that said he was going to be happy (or sad), so that he could decide in favor of (or against) Arthur's request. No such inference appeared, so as a last resort, REQUEST asks:

HOW KIND IS GEORGE BIRD?

1:VERY 2:SOMEWHAT 3:NOT VERY 4:NOT AT ALL \*2

Will he do it out of the goodness of his little heart? Yes. So REQUEST calls DO-MTRANS -- George has decided to tell Arthur where some honey is. Now you may or may not realize this, but George hasn't the faintest idea where there's any honey. In fact, there isn't any honey anywhere in the entire world -- yet. DO-MTRANS is built to handle this: If the storyteller has decided that George is going to tell Arthur where some honey is, then some honey will be created. But there's even more complexity afoot. We want to permit characters to lie to each other, and the decision whether to lie is made inside DO-MTRANS, which asks:

DOES GEORGE BIRD FEEL DECEPTIVE TOWARDS ARTHUR BEAR?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*1

He's going to lie. DO-MTRANS calls the creation-functions, but the results are stored only in George's memory, not in the real memory as well.

The picture procedure for creating honey specifies that the minimal environment for honey includes a beehive and a bee who owns the honey. Omitting the details, we get \*IVANBEE\* who owns \*HONEY\*3 which he keeps at home in \*BEEHIVE\*0 which is in the \*REDWOODTREE\* which is surrounded by \*GROUND\*1 and is located in \*VALLEY\*1. The physical locations really are created; the fictitious data is that the beehive is in the tree, that the bee and the honey are inside the beehive, and that the bee owns the honey: Only George "believes" these things. But he tells all them to Arthur. The inference-maker churns away with these new items. Arthur knows that George told him these things. But the inference-maker must know more about Arthur and George before it can say whether Arthur believes them all, so it asks:

DOES ARTHUR BEAR TRUST GEORGE BIRD?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

He believes them. He reacts to the news about honey: Honey is one of his foods, and he is hungry. Normally, this would produce a new goal: satisfy hunger. But satisfying his hunger is already part of his goal structure, so the new goal is ignored. It may seem odd for a character to acquire the goal of eating while pursuing the goal of eating, but this mechanism is intended to activate the goal if it wasn't already active, if he found out by accident where some food was.

So the inference-maker stops, DO-MTRANS returns to REQUEST which returns to the first call of DO-MTRANS which returns to TELL which returns to PERSUADE. The goal of the persuasion has been satisfied -- Arthur now thinks he knows where there's some honey -- so PERSUADE

returns to DKNOW which returns to DCONT. Arthur's goal structure was cleaned up along the way. As each planning procedure succeeded, the corresponding goal was removed from the goal structure. His current goals are to be not-hungry (zero on the hunger scale) and to own some honey. All this has gone towards achieving only the first precondition of DCONT, finding out where some honey was.

#### 11.21 BACK IN DELTA-CONTROL

**T**he first planbox for DCONT asks whether the desired object is free for the taking, that is, whether anyone owns it. Arthur thinks that Ivan owns it, so it isn't free for the taking, and we go to planbox 2.

At planbox 2, we consider tricking the owner, but this requires some character information.

HOW HONEST IS ARTHUR BEAR?

1:VERY 2:SOMEWHAT 3:NOT VERY 4:NOT AT ALL \*1

Forget that, we have a bear with scruples on our hands. The next planbox is very simple: Arthur will try to PERSUADE Ivan to give him the honey.

## 11.22 PERSUADING AGAIN

**T**he first planbox in PERSUADE is ASK, and in it we check some social relations.

DOES ARTHUR BEAR FEEL DECEPTIVE TOWARDS IVAN BEE?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

DOES ARTHUR BEAR FEEL COMPETITIVE TOWARDS IVAN BEE?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

DOES ARTHUR BEAR LIKE IVAN BEE?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

DOES ARTHUR BEAR THINK THAT IVAN BEE LIKES HIM?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

DOES ARTHUR BEAR THINK THAT IVAN BEE IS TRYING TO DECEIVE HIM?  
1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*4

Now you and George and I know that there's no such person as Ivan Bee, but Arthur doesn't, so it isn't as silly as it may sound to answer these questions. Like George, Ivan passes the test, so PERSUADE calls TELL -- to ask the question -- which calls DPROX -- to get him close enough to talk -- which calls DKNOW -- to find out where Arthur thinks Ivan is (\*BEEHIVE\*0) and where he himself is (\*GROUND\*0). Since those two locations aren't "close," DPROX calls DLINK which creates a border between \*MEADOW\*0 and \*VALLEY\*1. Then DPROX calls DO-PTRANS, and Arthur walks over to \*GROUND\*1. (Among the inferences from the PTRANS is the fact that George knows that Arthur is no longer at \*GROUND\*0, which is the first time we were able to make this inference.)

#### 11.23 THE LIE DETECTED

**B**ut when we test to see whether Arthur is "close" to Ivan Bee, memory says no. Since Ivan doesn't exist, nobody can be "close" to him.

Several things happen. First, Arthur now knows that Ivan isn't there after all. Next, DPROX checks memory to see whether Arthur knows who told him Ivan would be there. He remembers that it was George. (This is the first time we have used the inference that "A tells B to C" implies "C knows that A told B to C.") Last, the relationship between Arthur and George automatically changes (that is, without asking questions): Arthur now distrusts George, likes him somewhat less, and believes that George is trying to deceive him.

DPROX calls DKNOW again to find out where Ivan is, since he still wants to ask him for the honey. DKNOW calls PERSUADE again, but this time George does not pass the test for simply being asked to tell Arthur where Ivan is -- the change in the relationship has precluded asking for simple favors. So the next planbox in PERSUADE is tried: bargaining.

#### 11.24 BARGAIN

**T**he technique in bargaining is to offer to do something useful for the person you're bargaining with. Something is "useful" if it helps achieve a goal. Now Arthur doesn't know the details of George's goal structure, but he assumes that George has the samesame goals that all



birds do, and the bird-goals specified in the initial world knowledge are simply the physical sigma-state goals: hunger, thirst, rest, and sex. Part of the information about those goals includes what someone can do to achieve them for someone else. You can bring someone food to help him satisfy his hunger-goal, for example. Not all the goals have preconditions which others can provide. Resting is something you do by yourself; they can't help you with that (unless, for instance, you require pillows or sleeping potions).

Arthur decides to help George satisfy his hunger-goal by offering to bring him some bird-food, in this case, a worm. To make the offer, BARGAIN calls TELL. TELL calls DPROX, which eventually succeeds. Now back at the maple tree, Arthur asks George the question: "Will you tell me where Ivan is if I bring you a worm?" After this message is communicated and all its inferences asserted, DO-MTRANS detects that it is a special kind of request, a promise: Arthur is asking George to promise to tell him where Ivan is if Arthur brings him a worm.

#### 11.25 PROMISE

**T**he PROMISE procedure asks whether George is trying to deceive Arthur. Indeed he is, so George agrees but decides that when Arthur does bring him a worm, he won't tell him where the honey is. Instead, he will tell him that he's stupid. This decision for future action is implemented with a simple demon mechanism attached to the



inference-maker: A one-shot inference from "George knows that Arthur gave him a worm" will be "George plans to call Arthur stupid." (See Chapter 10.)

Had he not been trying to deceive Arthur, and had he felt some affection for him, George would have agreed to the bargain, and the one-shot inference from "George knows that Arthur gave him a worm" would have been "George plans to tell Arthur where some honey is." Otherwise, he would simply have refused. (Obviously, that's being overly simple -- he should go through the same kind of cogitations we saw in REQUEST.)

PROMISE returns to DO-MTRANS, which returns to TELL, which returns to BARGAIN, which now checks to see whether Arthur thinks that George is going to tell him where Ivan is. That's true, so Arthur now attempts to carry out his part of the bargain. BARGAIN calls DCONT for Arthur to get a worm. DCONT calls DKNOW for him to find out where a worm is, and DKNOW first checks memory to see whether Arthur knows where any worms are. Lo and behold, he does. Long ago, at the beginning of this story, we specified that Arthur was to know that \*WORM\*0 was over at the patch of ground, and Arthur hasn't forgotten this. DKNOW succeeds, and DCONT now asks whether Arthur thinks that the worm is free for the taking. Arthur doesn't know about anyone owning the worm, so he assumes it's free. So DCONT calls DPROX to get Arthur over to the patch of ground. DPROX calls DLINK to construct a route. We get a new valley between the meadow where Arthur is and the meadow where the worm is, and all the border points as well. The journey takes place.

Before having Arthur take the worm, DCONT checks whether Arthur now thinks anyone owns the worm -- he may have found that out by the end of the journey. But there's no such new information, so DCONT calls DO-ATRANS, and Arthur picks up the worm. DCONT succeeds.

BARGAIN now calls DPROX to get Arthur back to George, and there are no problems in the return journey. Finally, BARGAIN calls DO-ATRANS for Arthur to give the worm to George. He does, and there are lots of inferences:

1. George has the worm.
2. Arthur doesn't have the worm any more.
3. George knows that Arthur has given him the worm.
4. Arthur knows that he has given George the worm.
5. George knows that he has the worm.
6. Arthur knows that George has the worm.
7. George knows that Arthur doesn't have the worm anymore.
8. Arthur knows that he doesn't have the worm anymore.

In computing George's reaction to having a worm, we need to know how hungry he is, because having a food may cause a goal of hunger.

\*CONT\*REACT, the inference procedure for reacting to knowledge about possession, asks memory how long it's been since George ate anything,

but memory says that it hasn't seen George eat anything since the story began, so we get the question:

HOW HUNGRY IS GEORGE BIRD?

1:VERY 2:SOMEWHAT 3:NOT VERY 4:NOT AT ALL \*1

Since we decided that he's hungry, the inference is that George will plan to satisfy his hunger. Since George is making that inference, it becomes a new goal for him. The inference maker calls S-HUNGER (SIGMA-HUNGER), which is what started this whole story, eons ago. But this one is much easier. S-HUNGER asks memory whether George knows that he has any food, and of course, he does. So DO-INGEST is called, and George eats the worm. There are six inferences from that: George is no longer hungry, the worm no longer exists, George knows both those things, and Arthur knows both those things. When "\*WORM\*0 no longer exists" hits the memory, all references to \*WORM\*0 are removed (see Chapter 6). S-HUNGER has succeeded.

#### 11.26 THE DEMON WAKES

**T**he next inference we were working on was that George knows that Arthur has given him a worm. The demon inference is made. When George first told Arthur that Ivan Bee existed, this demon was created. It specified that when Arthur brought George a worm, George should react not with gratitude but with an insult. This demon was attached to the ATRANS reaction procedure, and once it had "fired," it was removed.

George tells Arthur that he's stupid (-6 on the intelligence scale). The inferences from that are that Arthur knows that George said he was stupid and that Arthur thinks that George believes that Arthur is stupid. The reaction-inference from this last one is a further deterioration in the trust and affection Arthur feels toward George. (Had it been +6 on the intelligence scale -- a compliment -- the relationship would have improved accordingly.)

#### 11.27 THREATEN

**B** ARGAIN failed. The next planbox is THREATEN -- Arthur is considering whether to bash George in the beak. He's mad enough, but wait:

DOES ARTHUR BEAR THINKS THAT GEORGE BIRD DOMINATES HIM?

1:A LOT 2:A LITTLE 3:NOT MUCH 4:NOT AT ALL \*2

That spoils it -- he's intimidated. PERSUADE fails, and Arthur can't find out where Ivan is, since he doesn't know anyone else to ask. We're back in TELL now -- the goal is still to ask Ivan to give Arthur the honey. The first method didn't work: Arthur couldn't ask Ivan directly. So he tries the second planbox: persuade someone else to ask Ivan. Sure enough, we're back in PERSUADE; this time Arthur's trying to persuade George to ask Ivan to give Arthur the honey. He knows he can't ask directly, so he tries bargaining again, offering George another worm. George agrees -- via the PROMISE mechanism -- although he's planning to use the same trick as before: If Arthur does bring him

a worm, he'll insult him again. To hold up his part of the bargain, Arthur now needs to get a worm. DCONT calls DKNOW -- he needs to find out where a worm is, but he doesn't know where any more worms are, so he tries to PERSUADE George to tell him. But the only thing he can think of to persuade him is to bring him a worm. But that's already in the goal structure, so that won't work.

#### 11.28 THE FALL OF ARTHUR

**H**e can't persuade George to tell him where a worm is, so he can't find out where a worm is to give to George, so he can't keep his part of the bargain, so George won't ask Ivan to give him some honey. There's no one else that Arthur can persuade to ask Ivan, so Arthur now considers offering Ivan a flower (bees eat flowers, remember?) if Ivan will give him some honey. In order to make the offer, he's got to go to Ivan, but he tried that before and it failed -- it's still sitting around in the goal structure. The only alternative is to persuade George to make the offer for him, but he can't persuade George to do anything -- that's also sitting around in the goal structure as an unachievable goal. So he can't persuade Ivan to give him any honey, and he's too honest to steal it from him, so he can't get the honey at all. And he lived hungrily ever after. The end.

## 11.29 PROGRAMMING DETAILS

### 11.29.1 DATA VS. PROCEDURE

**T**ALE-SPIN is written in MLISP, an ALGOL-like language which is translated into LISP (Stanford 1.6), and it runs in LISP. The data files are either LISP code or S-expressions processed by LISP functions.

The planning structures (e.g., DELTA-PROX) are implemented as MLISP functions. Much of the information about them, however, is represented as data. The fact that SIGMA-HUNGER requires DELTA-CONTROL of some food, for instance, is represented not only in the MLISP procedure for SIGMA-HUNGER, but also as information about HUNGER, one of the primitive states in Conceptual Dependency, and also as an inference from the conceptualization that someone is no longer hungry -- i.e., if that is true, then he must have eaten something, and that implies that he had some food.

Why represent the same information in different ways? It might be possible to design a single representation that could be interpreted in a number of ways, as a problem-solving procedure, as a set of inferences, and so on, but it's not clear that people actually have one body of information about hunger, say, which they "consult" in different ways. The context in which the information is accessed isn't simply an important issue. It's the only issue.

## 11.29.2 THE SYNTAX OF CD

The basic unit of information in the system is the Conceptual Dependency expression. The grammar of these expressions is simple. There are three productions, <CONCEPT>, <FORM>, and <MODIFIER>. A <CONCEPT> either has the structure (CONS <FORM> <MODIFIER>), or else is an atom whose value has that structure. A <FORM> is either an atom or a <MODIFIER>. A <MODIFIER> either is the atom NIL or has the structure (CONS atom (CONS <CONCEPT> <MODIFIER>)). The atom which is the CAR of a <MODIFIER> is called a role, and the <CONCEPT> which is the CADR of a <MODIFIER> is called the role filler. Thus, in the CD expression

```
((ACTOR *JOHN*
  IS (*LOC* VAL (*PROX* PART *NEWYORK*))))
```

the atom-concept \*JOHN\* fills the ACTOR role, the concept (\*PROX\* PART \*NEWYORK\*) fills the VAL role, and so on. These expressions are constantly being constructed, examined, changed, and disassembled. The sample expression above is the representation for "John is in New York," and would be constructed by passing the parameters \*JOHN\* and \*NEWYORK\* to the function ISAT, written in MLISP as:

```
ISAT ('*JOHN*', '*NEWYORK*');
```

The function ISAT is simple and obvious.

```
EXPR ISAT (ACTOR, LOK);
<<'ACTOR, ACTOR,
  'IS, <'*LOC*, 'VAL, <'*PROX*, 'PART, LOK>>>;
```

Similar construction functions are used for all the other states and



acts in CD.

### 11.29.3 PATH AND XPLD

Examining a particular part of an expression is done by the function PATH, described by Riesbeck in his thesis [22, page 76]. The path consists of a list of roles, each embedded in the filler from the previous role. In the example above, the (ACTOR) path leads to \*JOHN\*, the (IS VAL) path leads to (\*PROX\* PART \*NEWYORK\*), and the (IS VAL PART) path leads to \*NEWYORK\*. The PATH function is used to "interrogate" the data: given that X is the CD which represents where John is, the (IS VAL PART) path of X will tell us the name of the location, in this case, \*NEWYORK\*.

TALE-SPIN is constantly examining CD expressions, usually several parts of a single expression. With larger expressions, it becomes tedious (and inefficient) to call the PATH function many times on the same expression. Yet many inferences depend on large expressions. For example, "John knows that Mary thinks he's an idiot" is represented as:

```
((CON ((CON ((ACTOR *JOHN*  
              IS (*SMART* VAL (-9))))  
        IS (*MLOC* VAL (*CP* PART *MARY*))))  
  IS (*MLOC* VAL (*CP* PART *JOHN*))))
```

We would like to make the inference that John will think somewhat less of Mary. The pattern for matching this requires that certain role fillers match each other and that certain forms have certain values.



This prevents us from matching "John knows that Mary thinks that Bill is an idiot," "John knows that Mary thinks that he's very short," "John knows that Mary thinks he's brilliant," and so on, from which we may want to make other inferences, but not this inference. The test can be expressed in MLISP as:

```
IF PATH ('(CON CON ACTOR), X) = PATH ('(IS VAL PART), X)
  AND CAR PATH ('(CON CON IS), X) = '*SMART*
  AND CAR PATH ('(CON CON IS VAL), X) LESSP 0
  THEN [...make the inference...];
```

Instead, a function called XPLD is used which computes all the possible paths in a CD expression, assigning the values to variables whose names are built up from the paths they describe. The above code can be rewritten as:

```
IF !CONCONACTOR = !ISVALPART
  AND !CONCONIS[1] = '*SMART*'
  AND !CONCONISVAL[1] LESSP 0
  THEN [...make the inference...];
```

The point of worrying about this detail is that large expressions are common in the present work. The following sentence is the generator's straightforward translation of an event common to many of the stories TALE-SPIN has written:

```
TOM BEAR ASKS GEORGE BIRD WHETHER IF TOM BEAR GIVES A WORM TO
GEORGE BIRD THEN GEORGE BIRD WILL ASK IVAN BEE WHETHER IF TOM
BEAR GIVES A FLOWER TO IVAN BEE THEN IVAN BEE WILL GIVE THE
HONEY TO TOM BEAR.
```

The representation for that is:

```

((ACTOR *TOMBEAR*
  <=> (*MTRANS*)
    MOBJECT ((CON ((ACTOR *TOMBEAR*
      <=> (*ATRANS*)
        OBJECT *WORM*13
        TO *GEORGE BIRD*
        FROM *TOMBEAR*)
        TIME (TIME267))
      CANCAUSE ((ACTOR *GEORGE BIRD*
        <=> (*MTRANS*)
          MOBJECT ((CON ((ACTOR *TOMBEAR*
            <=> (*ATRANS*)
              OBJECT *FLOWER*1
              TO *IVANBEE*
              FROM *TOMBEAR*)
              TIME (TIME227))
            CANCAUSE ((ACTOR *IVANBEE*
              <=> (*ATRANS*)
                OBJECT *HONEY*2
                TO *TOMBEAR*)))
          MODE (QM4)
          TIME (TIME257))
            TO (*CP* PART *IVANBEE*))
            TIME (TIME256)))
        MODE (QM6)
        TIME (TIME269))
        TO (*CP* PART *GEORGE BIRD*))
        TIME (TIME268))

```

It needs a lot of unscrambling before any inferences can be made. The generator (see Chapter 12) also needs to examine various pieces of a conceptualization so that it can quickly determine the most succinct way of expressing an idea. Since inference-making, especially, is such a basic part of the simulation process, improving the efficiency of that code has significantly improved the performance and response time of the entire system.

## CHAPTER 12

### TELLING A STORY -- IN ENGLISH

#### 12.1 THE GENERATOR

The task of translating Conceptual Dependency expressions -- conceptualizations -- into English is an example of the "how-to-say" problem, and was the subject of Neil Goldman's thesis [12]. The existence of a first solution to a problem, even a partial solution, often simplifies later attempts to deal with the same or related problems. The generator (the program which produces an English translation) is modeled after Goldman's.

TALE-SPIN doesn't use Goldman's actual generator or the modified version which Walter Stutzman and Jerry De Jong have produced for the SAM system [30]. That program, called BABEL, is currently three-quarters the size of TALE-SPIN, and the two programs are too large to be run together on the system at Yale. Even running them separately and communicating via disk files proved too cumbersome. But more

important, TALE-SPIN and BABEL were not terribly well suited to each other: BABEL has many features, like automatic paraphrasing, that TALE-SPIN doesn't use, and TALE-SPIN requires some constructions, like certain modals, that are not implemented in BABEL. So I abandoned the idea of using BABEL and, instead, wrote MUMBLE. It's a quick 'n' dirty program, written in a day, and it many of its parts do not correspond to the way humans speak, but it produces adequate, if somewhat verbose, sentences. Best of all, it's one tenth the size of BABEL.

The translation is built up a word at a time. Tenses, modals, and infinitives are done when the verb is added. Articles (a, an, the, some) are added in a second pass through the sentence.

Most of the translations are straightforward. Some of the more interesting cases are listed below.

#### Warning

The nitty-gritty level of detail below may offend some readers.

#### 12.1.1 VERBS (ACTS)

If the CD to be expressed contains one of the primitive acts, we use the code associated with the particular act to do the translation. The code uses a simple discrimination net to find the best translation.

No paraphrases are included.

If the act is PTRANS, we need to decide whether to say "walk" or "fly." We do so by looking at the type of the ACTOR. Part of the information stored under the notion "human" is that humans commonly move their legs to get from place to place (if they are doing the moving themselves). This allows us to distinguish "walk" from "fly," for instance. The MANNER-role may tell us something about the speed of the PTRANS, in which case we would use "run," not "walk."

The English names for some locations are not very descriptive. The program may think of the ground below the oak tree as \*GROUND\*13, and that's sufficient for it to distinguish it from any other patch of ground, but it's not very useful to translate this as "the ground," since that doesn't distinguish it from any other patch of ground, or, in fact, from the generic "ground." In this case, we take the immediate physical context in forming the name to print for a particular piece of ground. Hence, "ground by the oak tree."

A new role added to Conceptual Dependency is ROUTE. The role-filler contains information from the map system (see Chapter 9), listing the points along the path of travel. Part of the information stored with generic locations -- valley, mountain pass, etc. -- is a preposition commonly used in expressing a journey through that location -- "across" the meadow, "over" the mountain pass, etc. These are added to the sentence. Thus:

SAM ADAMS WALKS FROM THE HOUSE TO THE GROUND BY THE REDWOOD

TREE BY GOING THROUGH A VALLEY ACROSS A MEADOW.

If the verb is ATRANS, the program makes the distinction between "give" and "take," depending on whether the ACTOR was the donor (FROM) or the recipient (TO).

MTRANS has several possible translations. If the MOBJECT is a question, we say "ask whether," as in "John asked Mary whether Bill was hungry." If the ACTOR is also the recipient of the information, we say either "see" or "remember," depending on whether this act has an instrument (INST) of ATTEND-EYES. Otherwise we simply say "tell that," as in "John told Mary that Bill was hungry."

We translate the act MBUILD as "decide." If the ACTOR of the MOBJECT is the same as the ACTOR of the entire conceptualization, we say "decide to" and use an infinitive construction for the MOBJECT, as in "John decided to go to New York." Otherwise, we say "decide that" and do not use the infinitive form, as in "John decided that Bill was a schmuck."

If the act is PROPEL, we say "fall" if the ACTOR is GRAVITY, "strike" if the OBJECT is HAND, or simply "propel" otherwise.

The act GRASP is either translated "hold" or "let go of," depending on whether the act was starting or terminating.

Scripts can be acts. \$SEX is translated "fool around." \$SLEEP is translated as "go to sleep" or, if memory says that the ACTOR has been

asleep recently, "go back to sleep."

Since the translations are simple, the question arises: Why not just use English in the first place? Why bother with a meaning representation?

First of all, the complexity of the translations is independent of the rest of the program. This generator could be made much more sophisticated without changing the planning structures, for example.

Second, generating the story is only a small part of the work done by the program. Very few of the conceptualizations that are passed along from one part of the system to another are ever expressed in English.

But more important than considerations of efficiency is the notion that while natural languages such as English may be well suited for expressing an idea, they are not particularly well suited to representing what the idea means. After all, natural language is only a method of communicating ideas from one person's head to another person's head. English is not what we know, it's what we use to express what we know. For that matter, there are things which are not even easily expressible in English, such as the difference between right and left (without "cheating" by referring to shirt buttons or subatomic particles.) So apart from their inherent problems, such as ambiguity, natural languages are inadequate tools for representing knowledge.



#### 12.1.2 STATES

**I**f the state is LOC, we use "where" if the location is unknown. Thus, "Is John in New York?" vs. "Where is John?" Also, "Mary asked Bill where John was" vs. "Mary asked Bill whether John was in New York."

The state MLOC presents several complications. If the idea which is known (CON) is a question, then the entire sentence is treated as a question. When the ACTOR of the CON is the same as the person who "knows" the CON, we convert from "long form" to "short form" (see Chapter 3), saying *"John is hungry"* instead of the more literal "John thinks he is hungry." This is the one case in which MUMBLE uses pronouns.

The phrases for specifying the degree are done in a uniform (uniformly inelegant) fashion. "John likes Mary not at all" could certainly be improved.

#### 12.1.3 CAUSALS

**M**UMBLE translates "<A> can cause <B>" as "If <A> then <B>" unless the previous word in the sentence was "whether," in which case it seems less stuffy to say "[whether] <B> if <A>." For example, "John asks Mary whether if he brings her some flowers then she will go out with him" sounds worse than "John asks Mary whether she will go out with him if he



brings her some flowers."

If thing "caused" is an increase in someone's happiness, we use "want to." This is how goals are represented in the program: "John wants to go home" is the euphonious version of the more literal "If John is home then he will become happier." Similarly, "want not to" is used with decreases in happiness. If the actors of the two parts match, then the subject is omitted: "John wants to go home," not "John wants John to go home." The infinitive form is always used with "want to."

The MLOC constructions, in particular, reiterate Goldman's point that the more context you examine, the better your translation can be. The completely blind translation for "John wants to go home" would be "John thinks that if John is at home then John will become happier." MUMBLE refers to memory to extend the context even further, improving the translations. "John went to New York" is much blander than "John returned to New York," but the second form can only be used in light of evidence, supplied by memory, that John was in New York at some previous time. MUMBLE knows how to say "John went back to New York," "John gave the apple back to Mary," and "John took the apple back from Mary." It also differentiates between "think" and "know": If John thinks X is true and X actually is true, then MUMBLE says "John knows that ..." Otherwise it says "John thinks that ..."

#### 12.1.4 VERB MODIFIERS

**T**he verb-builder is a very short procedure which needs to know the infinitive form of the verb, the tense, whether the sentence is a question, whether the subject is singular or plural, whether the sentence was to be negated, and whether to use an infinitive construction. That is sufficient to produce such constructions as "John might be able to go to New York" and "wouldn't John have wanted Mary to like him?" from their meaning representations.

#### 12.1.5 ARTICLES

**A**fter the sentence has been constructed, a second pass is made to insert articles before the nouns. The decision is based on properties associated with the nouns themselves. These properties are added when the person or object named by the noun is created.

The following rules are applied in order.

1. The property PRNAME indicates a proper name, like "Sam Adams." No articles are used with proper names.
2. All other nouns have the property ELEX, for "Eh lexeme." Some nouns are non-specific. When we say that Joe Bear wanted to find a beehive with some honey, we're not referring to any specific beehive or honey. Unspecific nouns which have the property MASS take the article some. The others take a or an.

3. If a noun has been explicitly mentioned already, it takes the article the. MUMBLE does not recognize implicitly mentioned nouns, as in "John went to a restaurant. The waitress took his order."
4. If a "new" noun is plural or has the property MASS, it takes the article some. Otherwise it uses a or an.

## 12.2 DIALOGUE

**F** laborating on a point made in the previous section, let's compare two examples.

1. a. TOM BEAR ASKS GEORGE BIRD WHETHER IF TOM BEAR GIVES A WORM TO GEORGE BIRD THEN GEORGE BIRD WILL ASK IVAN BEE WHETHER IF TOM BEAR GIVES A FLOWER TO IVAN BEE THEN IVAN BEE WILL GIVE THE HONEY TO TOM BEAR.
- b. GEORGE BIRD TELLS TOM BEAR THAT GEORGE BIRD WILL ASK IVAN BEE WHETHER IVAN BEE WILL GIVE THE HONEY TO TOM BEAR IF TOM BEAR GIVES A FLOWER TO IVAN BEE.
2. a. "George, would you ask Ivan if he'll trade me his honey for a flower? I'll give you a worm if you do."
- b. "Sure, Tom."

The first example is MUMBLE's translation of a conceptualization that occurred during an involved TALE-SPIN story. The second is a hand translation of that into dialogue. Assuming that the examples have the same meaning, and ignoring the lack of pronouns, why is the second example so much better? What would it take to produce it?

The "easy" part first: "Sure, Tom." When someone agrees to a specific request, that is, when his answer exactly matches the request, the translation can be abbreviated. (If we weren't doing dialogue, we could simply say that he agreed.) But it takes a memory system to know that there was a request; you cannot abbreviate sentence 1b to sentence 2b outside of the context of a request. In the example above, the request is explicit and immediately precedes the answer. It needn't be.

Big Bad Bart stalked into the saloon, lookin' mean. He saw Chicken Charlie sittin' with Esmerelda, who was Bart's girl. He walked up to Charlie, not sayin' a word. Charlie said, "Sure, Bart," and made himself scarce.

It's not hard to figure out what Charlie has agreed to, but it's hardly explicit. The abbreviation signifies that Charlie believes that a silent request was made, which is not to say that such a request was actually made. That is, Bart may not have intended that, and since he didn't vocalize his intention, we're not certain Charlie understood.

Let's consider the merits of sentences 1a and 2a, first from the standpoint of reading (parsing), and second, from the standpoint of expressing (generating).

Sentence 1a has little to recommend it from the standpoint of

parsing, except that it is self-contained: it's all there. If you can follow it at all, you've probably understood it. Sentence 2a, on the other hand, requires that we make a mid-course correction, because it's actually two sentences. The first sentence suggests that the ASK planbox is being used, that Tom is simply asking George to do him a favor, but we learn in the second sentence that it's actually the BARGAIN planbox, that some payment is involved. These are not very different ideas: the correction is more of a patch than a rewrite. So the tradeoff between what we might call the Henry James version (1a) and the two short sentences (2a) is a tradeoff between the tortuous and the briefly erroneous. The latter wins because people make such corrections all the time. (See Riesbeck [22] for a discussion of backup in resolving ambiguity.)

How about generating those sentences? Which is easier? The purpose of speaking is to be understood by a listener; the purpose of writing is to be understood by a reader. Written dialogue is a mixture, but it represents speech, so let's think of it that way for the moment. One difference in the way speech and text are understood is that text can be re-scanned, easily and arbitrarily often. Another is that many people can read faster than they can listen. This suggests that text can be more complicated than speech, that you'd lose money selling Hegel on an LP. So 1a is reasonably appropriate to its medium. But a two-sentence rewrite of 1a would be even better. How can we do that?

BABEL and MUMBLE always produce single-sentence translations, going

on the theory that parsimony is always preferable. While that's a good theory, they don't do too well with complicated text for which no single-sentence parsimonious translation is found. They simply iterate, blindly, stringing together translations of the sub-sentences. It would be better for them to detect such sentences early on, and to express each of them as two sentences.

What characterizes conceptualizations that need to be expressed in two or more sentences? Only that the person or program lacks a way of expressing it in few words, and that's independent of the form of the conceptualization, dependent on the language itself. For instance, there are two instances of bargaining in sentence 1a, "I'll ATRANS if yourade" is a handy abbreviation for the second one, but there isn't an abbreviation for the first one.

Looking at dialogue has shown us three things. First, dialogue often contains abbreviations, and they can be neither understood nor produced without reference to some context, which adds support to the claim that the more context you know, the better your translation can be. Second, the goal of parsimony is often abandoned in dialogue. While people may write terse prose, they may speak in more long-winded phrases, due to the difference in speeds of reading and listening. Third, dialogue contains many examples where a single conceptualization is best expressed in two sentences. While the first sentence may be misunderstood until the second sentence is read, this brief error is easily corrected in the same way that people resolve ambiguities.

## CHAPTER 13

### THE END

#### 13.1 USING PEOPLE AS MODELS OF STORYTELLERS

**W**hat has it gained us to have used people as models of storytellers?

First, it means that much of our work, ironically, has nothing whatsoever to do with stories, which is good because we are interested in lots of other things which people do besides tell stories.

Second, it enables us to rely on our knowledge and intuition about people as opposed to relying on our knowledge about computers and what best suits them. Using square roots where people do not may produce faster programs but may also lead to conflicts in theory. Using square roots where we don't know what people do is perfectly acceptable. There's no hope of solving all the problems right away, so we take care to do the right thing when we know what the right thing is.



Third, it gives us a new outlook, if you will, on psychology. Storytelling is a combination of how people behave and how they communicate. What happens in the story tells us about how people behave; the way in which the story is told -- what particular things happen, what things are included or left out -- tells us about communication. People never say every little thing that happens. There isn't time. Since communication is the task of getting ideas from one person's head to another's, the structure of communication -- why things are expressed in a particular form -- is an important subject to study if we're interested in what those ideas really are.

### 13.2 EXTENSIONS

**A**s I discuss extensions to the present system, I'd also like to make it clear what the present system is, because there's been a great deal of theory discussed side-by-side with actual implementation, and while that's a legitimate expository style -- specific examples presented in the context of a general theory -- it can be misleading. Anyone who has read Drew McDermott's grumpy/funny exhortation for honesty in AI [16] will understand.

I'll start with the delta-acts, since they're so important. DELTA-PROX, as advertised in Chapter 3, has three planboxes which are implemented, and four others that aren't. Of those four, the last two require no additions to the program. One of the other two requires



transportation-system scripts (cars, buses) which I haven't looked at yet. Scripts in general have not been incorporated into TALE-SPIN except in a very simple fashion (e.g., \$SLEEP). They're crucial to a realistic problem-solver -- we may use scripts more than anything else -- but they're so developed that they're uninteresting: not great story material. Much work has been done with scripts as aids in understanding, however, and it would be great to use the same scripts in building a script-based problem-solver, even (especially?) if it revealed that you need different representations for understanding and problem-solving.

Another planbox in DELTA-PROX says that X and Y (suppose it's you and I) should meet in some common location. That's a good idea, but it requires some hard work. How do I know what places you know how to get to? Perhaps I could give you directions. Of course, since TALE-SPIN doesn't have any remote-communication systems (e.g., telephones) either, I'd have to go to you to give you directions, which is silly since getting near you is already my goal. Even with telephones, my directions would be how to walk to the common location, since there are no transportation systems. All in all, I'd say that work on this planbox should follow work on transportation and communication.

The map theory works well for finding routes, and the blueprints have been made easy to write -- there's a package of blueprint-reading functions which does all the work. But the map theory has some gaps. For example:

1. Separate maps should be maintained for each actor, as is the policy for other information. My map of Utah is blank; presumably, a Utah resident's isn't. Faulty maps should be made possible.
2. Maps now have connectedness but no direction. Going through three rooms, up a stairway, and into a hall is very much like going through five rooms all on the same floor.
3. Maps grow as the story requires, but no part of a map changes status with time. "Joe locked the door" should have the obvious effect on the map.

The current implementation of DELTA-KNOW has 4 planboxes, although the first one ("Standard Reference") isn't used yet. The real problem with DELTA-KNOW is that it should have a planbox for each problem domain, separate routines for answering each of these questions:

1. Who owns that house?
2. What is the capital of Connecticut?
3. What is the square root of sixteen?
4. What is the dominant of C?
5. What is the plural of mouse?
6. What is the second possible isotope of hydrogen?

So working on DELTA-KNOW is no harder than working on the domains themselves. If only that were easy.

*The current implementation of DELTA-CONTROL has four solution methods. Additional ones would include scripts -- very common ones, like "department store" or "restaurant" -- and plans to motivate the present owner(s) to hand over control, getting them to like you and so forth, not quite the same as trickery. DELTA-CONTROL, like the other delta-acts, should eventually be split into problem domains, recognizing the difference between obtaining control of an estate, a million dollars, an apple, a car, a slave, and a corporation. The simple methods, like "persuade the owner to give it to you," apply to some of those but not to others.*

*There are also the "negative" planning structures; DELTA-NEG-PROX was discussed and implemented. But there should also be a DELTA-NEG-CONTROL (getting rid of something) and a DELTA-NEG-KNOW (becoming convinced that something is false that you thought was true).*

*An area that needs more study and would significantly improve the world model is that of what we might call the mental sigma-states. TALE-SPIN has models of the physical sigma-states, and it knows some things about personality and social relationships, but not enough. The intellectual and emotional states form an enormous body of knowledge. It's plausible that the number of ways of getting from one place to another -- the number of planboxes in DELTA PROX -- is not huge. It's also hard to believe that there are all that many ways of getting rest. But how many ways are there to arouse curiosity? What does it mean to be well-adjusted? Why do people enjoy re-reading some books they like*

and not others?

The goal mechanisms discussed in Chapter 5 are only partially implemented. The very earliest version of TALE-SPIN had one goal stack for the entire system. With multiple actors who could interfere with each other, that got split into separate stacks. When the program started generating stories where characters kept trying goals that had previously failed, each stack turned into two structures, "current" and "failed" goals. When characters made promises, the demons appeared. And that's where the implementation stands. The theory which I described goes on to show a need for keeping large sets of goals around for each character, in three categories: "being pursued," "being preserved," and "failed." The problem solver would evaluate the worth of a goal by seeing how well it fits in with all these goals.

Some of the information about plans is available as data (preconditions, for example), but we need more. A fourth category of goals is "prevention goals," which are achieved by counter-plans, i.e., plans to thwart plans. So for every method we know about to get Joe Bear some honey, there may be a way to prevent it. Henry Bee, the owner of the honey, would be likely to use such a counter-plan.

In Chapter 7 I presented a theory of stories and two methods of storytelling. The bottom-up method is simply to let the reader control the simulator. The reader's tendency is to create lots of problems, not to make everything simple. That is, he relies on his instincts about stories, consciously or not, to make the simulation interesting.

The top-down approach is much harder to implement because there's no way to pre-compute all the details in the world that must be set before the simulation begins, and the top-down stories that have been done have more detail built into them than I would have liked. One way to "solve" this problem is to represent everything, especially the planning structures, as data, in the form of theorems, so that you can "prove" that the story's going to behave the way you want it to. But no human writer does that. Instead, we need to know what some of the details are that have to be rigged. When the simulation starts and we come across a decision we haven't answered yet, we should have some idea what the answer should be, unless we know it makes no difference, in which case, any answer will do. The temptation is to use backtracking -- assume the answer is "yes" and continue the simulation. If the story doesn't come out right, back up, say "no," and proceed. Unfortunately, the decisions are not all yes/no decisions. But more important, it's hard to imagine a human writer doing that blindly. He's more likely able to compute, not simulate, what the relevant effects of the decision will be, and decide on that basis. The point is here is much like the point I tried to make in Chapter 4: The goal is to have enough knowledge already in the system so that you don't have to resort to blind technique.

What would improve TALE-SPIN's ability to write stories? First, more problem-solving techniques. It has DELTA-PROX, DELTA-CONTROL, DELTA-KNOW, TELL, and DELTA-NEG-PROX, along with the PERSUADE package (ASK, INFORM REASON, BARGAIN, and THREATEN), special procedures for

REQUESTs and PROMISEs, and the sigma-states related to hunger, thirst, rest, and sex. To tell stories about interference among various characters, it needs the other "negative" delta-acts, mental sigma-states (for more sophisticated kinds of persuasion), and goal mechanism which includes the features described in Chapter 5. This last item is very important, since it removes some of the single-mindedness that characterizes the actors in the current TALE-SPIN stories.

To get more fables, we could add arbitrarily many new procedures to tell stories with particular morals, which would use built-in specifications for rigging the world model in advance. But more general techniques are needed, and may result from increasing the predictive power of the planning procedures, enabling the storyteller to "think ahead" much more easily than it can now.

While expanding the world model will certainly enhance the program's capabilities as a storyteller, so would the ability to use several linguistic styles. Some aspects of that will come from being able to summarize whole planning procedures in a few words -- there's a long-winded way to say "John wanted revenge," which would include a literal expansion of that phrase's meaning, and until the generator can communicate better with the problem solver, we're stuck with long-winded versions. But beyond that, there are intrinsic stylistic variations. A given story may be expressible in several styles: terse, prolix, humorous, allegorical, on and on. We haven't begun to tackle those problems. Characterizing what makes a story funny is a related problem;

we've barely managed to make stories coherent.

We also need to find a better way to handle the personality states and relationships. Doing it one scale at a time, either as input from the reader or as output from the generator, is unsatisfactory, tedious. It's conceptually correct to break down such things into some set of primitives, but the system should not have to converse with the user in terms of those primitives. Just as the generator can take several primitive states and acts at a time and produce one succinct way of expressing them all, so should it be able to describe relationships that way. "John and Mary have been married 34 years" is so much nicer than "John loves Mary and John thinks that Mary loves him and Mary loves John and John trusts Mary completely and ...."

### 13.3 STORIES

What have we learned about stories and storytelling? Stories must be coherent -- causally connected -- on many levels, and they must be interesting. We view interest as a product of conflict and stories as problems, therefore, with a principal problem domain which we call the domain of interest. In the top-down mode of storytelling, the domain is chosen after (and independent of) the main idea or point of the story, the theory being that almost any domain can be used to illustrate the point. In the program, the reader chooses the point to the story, and the program then rigs the world model in such a way that when the



characters behave rationally, an interesting story results, making the point that the reader wanted. In the bottom-up mode, the reader chose the initial setting, including the problem which sends the characters into action.

We discussed two levels of style, linguistic style and plot style, and we view each as being associated with a very different problem domain, but our theories on style are not part of the actual program.

The basic approach has been storytelling by simulation. What is the function of simulation itself? The process of making inferences about hypothetical events is a simulation, and the characters in TALE-SPIN do that all the time in figuring out ways to deal with each other. "Well, if I say I like his singing, then he'll probably like me for that, so when I then ask him to sing, he will, and the cheese will fall down, and ... Yeah!"

Could the characters themselves ever use the simulation as a means of storytelling? They certainly don't now, but there's no reason they shouldn't be able to. Storytelling, when you're trying to communicate some particular idea to someone, is clearly a planbox under TELL. For Aesop, it was probably even a script! Anyone who writes a story is simulating a world, and there's no reason why writers shouldn't exist in the simulated world as well. All tales-within-tales have that feature. John Barth has written a consciously tortuous story, "Dunyazadiad" [3], which is tree-structured. Each descendant node is a story within its parent node-story. The tree goes to depth five (at least -- it's hard



to count), but worse yet, one of the third-level substories seems identical to the root node. That is, a few levels down, we begin to hear a story about John Barth writing "Dunyazadiad." A recursive story! And there's an amusing science fiction book by Daniel Galouye [11] in which the characters in a simulated world discover they are being simulated and manage to escape into the higher world, which is, of course, itself a simulation. Galouye stops at three levels, but to consider the general case -- n levels of simulation -- or the ultimate case -- infinitely many levels of simulation -- produces a curious entailism.

Finally, we claim that there can be no adequate model of *stories* or *storytelling* without a model of the real world, not simply for the sake of completeness of detail, but also because the structure of stories is determined by real-world knowledge. That is, we can't talk about stories without also talking about human conflicts and their resolutions, which brings in problem solving. Our knowledge about what must be included in a story and what we can leave out depends on our knowledge about the *inferences* that people make. While there is story-specific information -- expectations we make when we know we're reading a story -- it must be integrated with our real-world knowledge in order to produce an adequate model of storytelling.

#### 13.4 WORLD KNOWLEDGE

**W**e've learned two principal things about world knowledge. First, we know that as the forms of knowledge vary from one domain to the next, so do their representations. World knowledge does not all look the same, and instead of looking for one way to represent everything, we should look for the representation which is most appropriate to the domain, the representation which best corresponds to the way in which people think of each body of knowledge. Second, we know that different knowledge representations can exist for the same object and that the choice among representations is determined by the context from which the information is being accessed. A newspaper is a source of information, a fly-swatter, and an historically interesting social force, depending on how you look at it. The question is not, "Should information exist as data or as procedures?" but rather "when is it data and when is it a procedure?" because it's often both. Even though semantic memory may be a byproduct of episodic memory -- you know that berries are food because you've eaten berries -- the "data" aspect of semantic memory still exists -- after a while, you don't need to remember whether you've eaten berries in order to remember that they're food.

#### 13.5 THE METANOVEL REVISITED


**O**ne of the unique features of the proposed metanovel we discussed in the Preface is that it would produce stories more complicated than

people could, which means that we will have to depart from the human model. But where? When? The implementation of the world model will have to permit much more flexibility than we now need. Reversing the story -- backtracking to a decision point, changing the decision, and restarting -- is perhaps the best example of a problem which has no counterpart in the "natural" world of human storytellers. It is an idea that really comes from computer science, not literature. Fortunately, there are programming languages like MICRO-PLANNER and CONNIVER that have already included some of the system-level features required by this idea, and the timesharing metanovel can certainly draw on current programming technology in operating systems to handle scheduling and consistency (literally reader/writer) problems. Sharing data segments (the memory of past and present events) is no theoretical challenge. The major technical difficulty may simply be the size of the system. TALE-SPIN runs in 75K on a PDP-10, but just barely; response is much better at 100K. Add a parser to the front end, or even a simple controller for more than one reader, and the size increases substantially.

But programming languages may well become advanced enough to implement the system, and memory may well become fast and cheap enough to support the system, long before we've solved the really hard problems of the metanovel. For some of these problems, such as linguistic style, we can rely on our present understanding of stories. But some of the problems, such as knowing how to regulate interference between two readers who control one set of characters, are completely foreign. Even

a multi-user version of TALE-SPIN would run into some of these problems, so it may not be too soon to begin working on them if we take that as our first task and worry about how several users would share control over events in the world of bears and bees. But these are new problems, not properly within the domain of AI since they are problems about humans - how the human readers interact -- but also are not problems about humans, since they don't correspond to problems which humans already know how to solve.

#### 13.6 PLANNING AND PROBLEM SOLVING

 ur approach is to separate problems into domains, each consisting of a set of representational primitives, problems expressed in terms of those primitives, and procedures for solving those problems. If we're modeling how humans solve problems, we've got to cope with the fact that people use radically different problem-solving techniques from one domain to the next. The features in common to all problems -- preconditions, actions, side-effects, and so on -- don't help us find the right representations. Indeed, those who believe that that kind of superficial similarity indicates a fundamental similarity about the domains themselves, and that a uniform representation is possible, even desirable, resort to theorem-proving as a means of solving problems. Predicate calculus provides an excellent representation for expressions in mathematical logic, but using it to represent "John sees Mary" is unnatural. Conceptual Dependency provides an excellent way of

representing "John sees Mary," but using it to represent "Boston is east of Chicago" is equally unnatural. The map representation which is natural for "Boston is east of Chicago" fails miserably for "Garth is an archaic synonym for garden," and so on.

Trying to make all the domains look the same is a waste of time. The important tasks are finding individual characteristics and trying to model how one domain communicates with another, for this is a truly phenomenal talent of humans, that they understand so many domains and can coordinate them with such apparent ease. Integration -- this process of combining models -- itself requires a theory. Working on integration is a difficult and frustrating task. We spend so much time trying to isolate a particular area -- how people figure out routes, for example -- and then we have to spend more time putting it back in, incorporating it into the total mode model -- how people use the routes they've figured out. A wrong choice may work for a while, but it can also cause problems which we cannot foresee. The "horror stories" of Chapter 8 illustrate how "reasonable" additions can lead to inconsistencies much later.

From a programming standpoint, integration means diffusion of code. There is an explicit procedure for SIGMA-HUNGER, but there are bits and pieces of what we might call SIGMA-JOY all over the place, and with reason: It's hard to believe that people can list the ways of becoming happier (SIGMA-JOY) in the same way that they can list the ways of satisfying their need for food (SIGMA-HUNGER).

Boundaries between areas of knowledge may be fuzzy, but they should be defined on the basis of what humans are like, not what computers are like. There are any number of possible computer models, but to compare them in computer terms misses the point of AI entirely. The only important criterion is the success with which they model man's endless source of fascination and awe, man.

## APPENDIX

### MORE EXAMPLES

There are four stories generated in mode 2 (omitting most of the resultant inferences). Paragraphs have been added here to indicate separate stories that use the same set of characters and locations.

The first story is the 'rescue' example, unsuccessful variations of which have been described elsewhere in the thesis. No information regarding the social relationships of the characters was needed by the program; Wilma, like all the characters, is inspired to rescue anyone in danger of death. Both George and Wilma recognize that George will drown if he stays in the water.

\*\*\* I \*\*\*

ONCE UPON A TIME GEORGE ANT LIVED NEAR A PATCH OF GROUND. THERE WAS A NEST IN AN ASH TREE. WILMA BIRD LIVED IN THE NEST. THERE WAS SOME WATER IN A RIVER. WILMA KNEW THAT THE WATER WAS IN THE



RIVER. GEORGE KNEW THAT THE WATER WAS IN THE RIVER. ONE DAY WILMA WAS VERY THIRSTY. WILMA WANTED TO GET NEAR SOME WATER. WILMA FLEW FROM HER NEST ACROSS A MEADOW THROUGH A VALLEY TO THE RIVER. WILMA DRANK THE WATER. WILMA WAS NOT THIRSTY.

GEORGE WAS VERY THIRSTY. GEORGE WANTED TO GET NEAR SOME WATER. GEORGE WALKED FROM HIS PATCH OF GROUND ACROSS THE MEADOW THROUGH THE VALLEY TO A RIVER BANK. GEORGE FELL INTO THE WATER. GEORGE WANTED TO GET NEAR THE VALLEY. GEORGE COULDN'T GET NEAR THE VALLEY. GEORGE WANTED TO GET NEAR THE MEADOW. GEORGE COULDN'T GET NEAR THE MEADOW. WILMA WANTED GEORGE TO GET NEAR THE MEADOW. WILMA WANTED TO GET NEAR GEORGE. WILMA GRABBED GEORGE WITH HER CLAW. WILMA TOOK GEORGE FROM THE RIVER THROUGH THE VALLEY TO THE MEADOW. GEORGE WAS DEVOTED TO WILMA. GEORGE OWED EVERYTHING TO WILMA. WILMA LET GO OF GEORGE. GEORGE FELL TO THE MEADOW. THE END.

The two relationships states at the end are demon inferences -- George would have reacted that way to anyone who PTRANSed him out of the water.

The second story uses SIGMA-SEX to motivate the characters. An elaborate blueprint for a house (see Chapter 9) was used. Each of the two characters starts off sitting in their living rooms, which, unfortunately, aren't close to the front doors, so that the blow-by-blow description of the trips from one house to another are excessive.



\*\*\* 2 \*\*\*

ONCE UPON A TIME JOE NEWTON WAS IN A CHAIR. MAGGIE SMITH WAS IN A CHAIR. MAGGIE KNEW THAT JOE WAS IN THE CHAIR. ONE DAY MAGGIE WAS HORNY. MAGGIE LOVED JOE. MAGGIE WANTED JOE TO FOOL AROUND WITH MAGGIE. MAGGIE WAS HONEST WITH JOE. MAGGIE WASN'T COMPETITIVE WITH JOE. MAGGIE THOUGHT THAT JOE LOVED HER. MAGGIE THOUGHT THAT JOE WAS HONEST WITH HER. MAGGIE WANTED TO ASK JOE WHETHER JOE WOULD FOOL AROUND WITH MAGGIE. MAGGIE WANTED TO GET NEAR JOE. MAGGIE WALKED FROM THE CHAIR ACROSS A LIVING ROOM DOWN A HALL VIA SOME STAIRS DOWN A HALL DOWN A HALL THROUGH A VALLEY DOWN A HALL DOWN A HALL VIA SOME STAIRS DOWN A HALL ACROSS A LIVING ROOM TO THE CHAIR. MAGGIE ASKED JOE WHETHER JOE WOULD FOOL AROUND WITH MAGGIE. JOE LOVED MAGGIE. JOE TRUSTED MAGGIE COMPLETELY. JOE DIDN'T HAVE MUCH INFLUENCE OVER MAGGIE. JOE REMEMBERED THAT JOE LOVED MAGGIE. JOE WALKED FROM THE CHAIR ACROSS THE LIVING ROOM DOWN A HALL VIA SOME STAIRS DOWN A HALL ACROSS A BEDROOM TO HIS BED. MAGGIE WALKED FROM THE CHAIR ACROSS THE LIVING ROOM DOWN THE HALL VIA THE STAIRS DOWN THE HALL ACROSS THE BEDROOM TO JOE'S BED. JOE FOOLED AROUND WITH MAGGIE. JOE BECAME HAPPIER. MAGGIE BECAME HAPPIER. JOE WAS NOT HORNY. JOE THOUGHT THAT MAGGIE WAS NOT HORNY. JOE WAS WIPED OUT. JOE THOUGHT THAT MAGGIE WAS WIPED OUT. MAGGIE THOUGHT THAT JOE WAS NOT HORNY. MAGGIE WAS NOT HORNY. MAGGIE THOUGHT THAT JOE WAS WIPED OUT. MAGGIE WAS WIPED OUT. MAGGIE WANTED TO GET NEAR HER BED. MAGGIE WALKED FROM JOE'S BED ACROSS THE BEDROOM DOWN THE HALL VIA THE STAIRS DOWN THE HALL ACROSS THE LIVING ROOM DOWN THE

HALL VIA THE STAIRS DOWN THE HALL DOWN THE HALL THROUGH THE VALLEY  
DOWN THE HALL DOWN A HALL VIA SOME STAIRS DOWN A HALL ACROSS A  
BEDROOM TO HER BED. MAGGIE WENT TO SLEEP. JOE WENT TO SLEEP. THE  
END.

The least Joe could have done would be to let poor Maggie sleep in  
his bed. As it was, SIGMA-REST, which was invoked because Maggie was so  
tired, simply specified that Maggie should get to her own bed, no matter  
where she was.

In the next story, we see an example of DELTA-NEG-PROX, which is  
now used in the STEAL planbox of DELTA-CONTROL -- when you want to steal  
something, get the owner away from it. That can be done using the  
PERSUADE package, as we saw in section 6.1. But it can also be done  
more directly, as we now see.

\*\*\* 3 \*\*\*

ONCE UPON A TIME LULU BEAR LIVED IN A CAVE. THERE WAS A NEST IN A  
MAPLE TREE. PEGGY BIRD LIVED IN THE NEST. LULU KNEW THAT PEGGY  
WAS IN HER NEST. ONE DAY LULU WAS FAMISHED. LULU WANTED TO GET  
SOME HONEY. LULU WANTED TO FIND OUT WHERE THERE WAS SOME HONEY.  
LULU LIKED PEGGY. LULU WANTED PEGGY TO TELL LULU WHERE THERE WAS  
SOME HONEY. LULU WAS HONEST WITH PEGGY. LULU WASN'T COMPETITIVE  
WITH PEGGY. LULU THOUGHT THAT PEGGY LIKED HER. LULU THOUGHT THAT  
PEGGY WAS HONEST WITH HER. LULU WANTED TO ASK PEGGY WHETHER PEGGY  
WOULD TELL LULU WHERE THERE WAS SOME HONEY. LULU WANTED TO GET  
NEAR PEGGY. LULU WALKED FROM HER CAVE DOWN A PASS THROUGH A VALLEY

ACROSS A MEADOW TO THE GROUND BY THE MAPLE TREE. LULU ASKED PEGGY WHETHER PEGGY WOULD TELL LULU WHERE THERE WAS SOME HONEY. PEGGY LIKED LULU. PEGGY TRUSTED LULU COMPLETELY. PEGGY DIDN'T HAVE MUCH INFLUENCE OVER LULU. PEGGY WAS INDEBTED TO LULU. LULU TRUSTED PEGGY COMPLETELY. PEGGY WAS VERY GENEROUS. PEGGY WAS HONEST WITH LULU. THERE WAS A BEEHIVE IN A REDWOOD TREE. MAGGIE BEE LIVED IN THE BEEHIVE. THERE WAS SOME HONEY IN MAGGIE'S BEEHIVE. PEGGY TOLD LULU THAT THE HONEY WAS IN MAGGIE'S BEEHIVE. PEGGY TOLD LULU THAT MAGGIE HAD THE HONEY. PEGGY TOLD LULU THAT MAGGIE WAS IN HER BEEHIVE. LULU WAS USUALLY HONEST. LULU WANTED MAGGIE TO GIVE LULU THE HONEY. LULU WAS INCLINED TO LIE TO MAGGIE. LULU DISLIKED MAGGIE. LULU DIDN'T HAVE MUCH INFLUENCE OVER MAGGIE. LULU DECIDED THAT MAGGIE WOULDN'T GIVE LULU THE HONEY. LULU WANTED MAGGIE TO GET NEAR THE GROUND BY THE REDWOOD TREE. LULU WANTED TO GET NEAR MAGGIE. LULU WALKED FROM THE GROUND BY THE MAPLE TREE ACROSS THE MEADOW THROUGH A VALLEY TO THE GROUND BY THE REDWOOD TREE. LULU GRABBED MAGGIE WITH HER PAW. LULU LET GO OF MAGGIE. MAGGIE FELL TO THE GROUND BY THE REDWOOD TREE. LULU WANTED TO GET NEAR THE HONEY. LULU TOOK THE HONEY. LULU ATE THE HONEY. THE HONEY WAS GONE. LULU WAS NOT HUNGRY. MAGGIE THOUGHT THAT LULU WAS NOT HUNGRY. THE END.

The last example shows how the generator now expresses the story that was used in Chapter 2. While the events are almost the same, the names, being chosen at random, are all different. (The name-picker seems to have a predilection for "Wilma.")

\*\*\* 4 \*\*\*

ONCE UPON A TIME BETTY BEAR LIVED IN A CAVE. THERE WAS A BEEHIVE IN AN APPLE TREE. MAGGIE BEE LIVED IN THE BEEHIVE. THERE WAS SOME HONEY IN MAGGIE'S BEEHIVE. TOM SMITH WAS IN A CHAIR. THERE WAS A NEST IN A REDWOOD TREE. WILMA BIRD LIVED IN THE NEST. TOM KNEW THAT WILMA WAS IN HER NEST. TOM KNEW THAT MAGGIE WAS IN HER BEEHIVE. TOM KNEW THAT BETTY WAS IN HER CAVE. THERE WERE SOME CRANBERRIES NEAR A BUSH. THERE WAS A WORM NEAR A PATCH OF GROUND. BETTY KNEW THAT THE CRANBERRIES WERE NEAR THE BUSH. BETTY WAS NOT HUNGRY. TOM KNEW THAT THE WORM WAS NEAR THE PATCH OF GROUND. ONE DAY TOM WAS FAMISHED. TOM WANTED TO GET SOME BERRIES. TOM WANTED TO FIND OUT WHERE THERE WERE SOME BERRIES. TOM LIKED WILMA. TOM WANTED WILMA TO TELL TOM WHERE THERE WERE SOME BERRIES. TOM WAS HONEST WITH WILMA. TOM WAS VERY COMPETITIVE WITH WILMA. TOM DECIDED THAT WILMA MIGHT WANT TOM TO GIVE WILMA A WORM. TOM WANTED TO ASK WILMA WHETHER WILMA WOULD TELL TOM WHERE THERE WERE SOME BERRIES IF TOM GAVE WILMA A WORM. TOM WANTED TO GET NEAR WILMA. TOM WALKED FROM THE CHAIR ACROSS A LIVING ROOM DOWN A HALL VIA SOME STAIRS DOWN A HALL DOWN A HALL THROUGH A VALLEY ACROSS A MEADOW TO THE GROUND BY THE REDWOOD TREE. TOM ASKED WILMA WHETHER WILMA WOULD TELL TOM WHERE THERE WERE SOME BERRIES IF TOM GAVE WILMA A WORM. WILMA WAS INCLINED TO LIE TO TOM. WILMA WANTED TO TELL TOM THAT WILMA WOULD TELL TOM WHERE THERE WERE SOME BERRIES. WILMA WANTED TO GET NEAR TOM. WILMA TOLD TOM THAT WILMA WOULD TELL TOM WHERE THERE WERE SOME BERRIES. TOM THOUGHT THAT WILMA WAS HONEST

WITH HIM. TOM WANTED TO GET A WORM. TOM WANTED TO GET NEAR THE WORM. TOM WALKED FROM THE GROUND BY THE REDWOOD TREE ACROSS THE MEADOW THROUGH THE VALLEY ACROSS A MEADOW TO THE PATCH OF GROUND. TOM TOOK THE WORM. TOM WANTED TO GET NEAR WILMA. TOM WALKED BACK FROM THE PATCH OF GROUND ACROSS THE MEADOW THROUGH THE VALLEY ACROSS THE MEADOW TO THE GROUND BY THE REDWOOD TREE. TOM GAVE WILMA THE WORM. WILMA WAS FAMISHED. WILMA ATE THE WORM. THE WORM WAS GONE. TOM THOUGHT THAT WILMA WAS NOT HUNGRY. WILMA WAS NOT HUNGRY ANY MORE. WILMA TOLD TOM THAT TOM WAS STUPID. TOM DISLIKED WILMA. TOM DISTRUSTED WILMA. TOM CONTROLLED WILMA. TOM STRUCK WILMA. WILMA DIED. TOM DECIDED THAT WILMA WOULDN'T TELL TOM WHERE THERE WERE SOME BERRIES. TOM DISLIKED MAGGIE. TOM DISLIKED BETTY. TOM COULDN'T FIND OUT WHERE THERE WERE SOME BERRIES. TOM WANTED TO FIND OUT WHO HAD SOME BERRIES. TOM COULDN'T FIND OUT WHO HAD SOME BERRIES. TOM COULDN'T GET ANY BERRIES.

BETTY WAS FAMISHED. BETTY WANTED TO GET SOME BERRIES. BETTY WANTED TO GET NEAR THE CRANBERRIES. BETTY WALKED FROM HER CAVE DOWN A PASS THROUGH THE VALLEY ACROSS A MEADOW TO THE BUSH. BETTY TOOK THE CRANBERRIES. BETTY ATE THE CRANBERRIES. THE CRANBERRIES WERE GONE. BETTY WAS NOT HUNGRY. THE END.

# REFERENCES

- 1] Robert P. Abelson.  
Concepts for representing mundane reality in plans.  
In Bobrow and Collins [4], 273-310.
- 2] Matthew A. Appelbaum.  
Meta-symbolic simulation system (MESSY) user manual.  
Computer Sciences Technical Report 272, University of Wisconsin,  
Madison, 1976.  
Includes a forward on the history of MESSY by Sheldon Klein.
- 3] John Barth.  
Chimera.  
Random House, New York, 1972.  
Includes three novellas: Dunyazadiad, Perseid, and Bellerophoniad.
- 4] Daniel G. Bobrow and Allan Collins, editors.  
Representation and Understanding: Studies in Cognitive Science.  
Academic Press, New York, 1975.
- 5] Timothy C. Diller, editor.  
Proceedings of the 13th Annual Meeting of the Association of  
Computational Linguistics.  
American Journal of Computational Linguistics, 1975.  
Available from Sperry-Univac, St. Paul, Minnesota 55101.
- 6] Scott E. Fahlman.  
A planning system for robot construction tasks.  
Artificial Intelligence 5:1-50, 1974.
- 7] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson.  
Learning and executing generalized robot plans.  
Artificial Intelligence 3:251-288, 1972.

- 8] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson.  
Some new directions in robot problem solving.  
Machine Intelligence 7:405-430, 1972.
- 9] Richard E. Fikes and Nils J. Nilsson.  
STRIPS: a new approach to the application of theorem proving to  
problem solving.  
Artificial Intelligence 2:189-208, 1971.
- 10] Northrop Frye.  
Anatomy of Criticism.  
Princeton University Press, Princeton, New Jersey, 1957.
- 11] Daniel F. Galouye.  
Simulacron-3.  
Bantam Books, New York, 1964.
- 12] Neil M. Goldman.  
Computer Generation of Natural Language from a Deep Conceptual  
Base.  
PhD dissertation, Stanford University, 1974.
- 13] Gary G. Hendrix.  
Modeling simultaneous actions and continuous processes.  
Artificial Intelligence 4:145-180, 1973.
- 14] Sheldon Klein et al.  
Automatic novel writing: a status report.  
Computer Sciences Technical Report 186, University of Wisconsin,  
Madison, 1973.
- 15] Sheldon Klein et al.  
Modelling Propp and Levi-Strauss in a meta-symbolic simulation  
system.  
Computer Sciences Technical Report 226, University of Wisconsin,  
Madison, 1973.
- 16] Drew V. McDermott.  
Artificial intelligence and natural stupidity.  
SIGART Newsletter (57):4-9, April 1976.
- 17] James R. Meehan.  
Using planning structures to generate stories.  
In Diller [5], microfiche 33:77-93.
- 18] Allen Newell and Herbert A. Simon.  
Human Problem Solving.  
Prentice-Hall, 1972.



- 19] Georges Polti.  
The Thirty-Six Dramatic Situations.  
The Editor Company, Ridgewood, New Jersey, 1916.
- 20] Vladimir Propp.  
Morphology of the Folktale.  
University of Texas Press, Austin, 1968.  
This edition is an English translation.
- 21] Charles J. Rieger.  
Conceptual Memory: A Theory and Computer Program for Processing the  
Meaning Content of Natural Language Utterances.  
PhD dissertation, Stanford University, 1974.
- 22] Christopher K. Riesbeck.  
Computational Understanding: Analysis of Sentences and Context.  
PhD dissertation, Stanford University, 1974.
- 23] Earl D. Sacerdoti.  
Planning in a hierarchy of abstraction spaces.  
Artificial Intelligence 5:115-135, 1974.
- 24] Erik Sandewall, conference committee chairman.  
Proceedings of the 4th International Joint Conference on Artificial  
Intelligence.  
MIT AI Lab, Cambridge, Massachusetts, 1975.
- 25] Roger C. Schank.  
Conceptual Information Processing.  
American Elsevier, New York, 1975.  
Includes contributions by Neil M. Goldman, Charles J. Rieger, and  
Christopher K. Riesbeck.
- 26] Roger C. Schank.  
The structure of episodes in memory.  
In Bobrow and Collins [4], 237-272.
- 27] Roger C. Schank and Robert P. Abelson.  
Scripts, Plans, and Goals: The Elements of Understanding.  
To be published by Lawrence Erlbaum Associates, Hillsdale, New  
Jersey, 1977.
- 28] Roger C. Schank and Robert P. Abelson.  
Scripts, plans, and knowledge.  
In Sandewall [24], 151-158.
- 29] Roger C. Schank and Bonnie L. Nash-Webber, editors.  
Theoretical Issues in Natural Language Processing (TINLAP).  
Association of Computational Linguistics, 1975.  
Available from ACL, 1611 North Kent Street, Arlington, Virginia  
22209.



- 30] Roger C. Schank and the Yale AI Project.  
SAM -- a story understander.  
Computer Science Department Research Report 43, Yale University,  
1975.
- 31] Gerald Jay Sussman.  
A Computational Model of Skill Acquisition.  
PhD dissertation, Massachusetts Institute of Technology, 1973.  
Published by the MIT AI Lab as AI TR-297.
- 32] Mary-Claire van Leunen.  
Scholarly Prose.  
To be published by Alfred A. Knopf, New York, 1977.
- 33] Joseph Weizenbaum.  
Computer Power and Human Reason.  
W. H. Freeman, San Francisco, 1975.
- 34] Myron Wish.  
Comparisons among multidimensional structures of interpersonal  
relations.  
Unpublished manuscript, Bell Laboratories, Murray Hill, New Jersey,  
1975.
- 35] Myron Wish.  
Perceived dimensions of interpersonal relations.  
To appear in the Journal of Personality and Social Psychology,  
1976.

